PATENT ABSTRACTS OF JAPAN

(11)Publication number : **2002-149464**
(43)Date of publication of application : **24.05.2002**

(51)Int.Cl. G06F 12/00

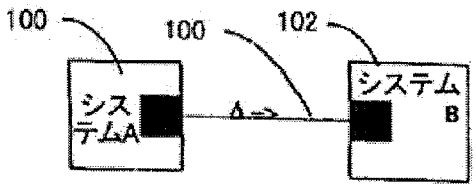
(21)Application number : **2001-287834**
(22)Date of filing : **17.08.2001**

(71)Applicant : **FUSIONONE INC**
(72)Inventor : **MULTER DAVID L**
GARNER ROBERT E
RIDGARD LEIGHTON A
STANNARD LIAM J
CASH DONALD W
JOSEPH ROBERTSON

(30)Priority
Priority number : **2000 641028** Priority date : **17.08.2000** Priority country : **US**

(54) BASE ROLLING ENGINE FOR DATA TRANSFER AND SYNCHRONIZATION SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To synchronize personal information among various types of devices.
SOLUTION: A first data package is provided with an identification number, an action, and a plurality of fields. A second data packet is provided with a second transaction following the first transaction. The second transaction is provided with an identification number, action, and fields having attributes. A base rolling engine decides whether or not the identification number of the second transaction corresponds to the identification number of the first transaction, and decides whether or not the identification number of the second transaction corresponds to one of the fields of the first transaction. When the identification number of the first transaction corresponds to the identification number of the second transaction, and the field of the second transaction corresponds to one of the fields of the first transaction, a first data package and a second data package are combined. The combined package is replaced with the second data package, and the first data packet is erased.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－149464
（P2002－149464A）
(43)公開日　平成14年５月24日(2002.5.24)

(51)Int.Cl.[7]	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　12/00	５３３	Ｇ０６Ｆ　12/00	５３３Ｊ　５Ｂ０８２

審査請求　未請求　請求項の数６　ＯＬ　外国語出願　（全111頁）

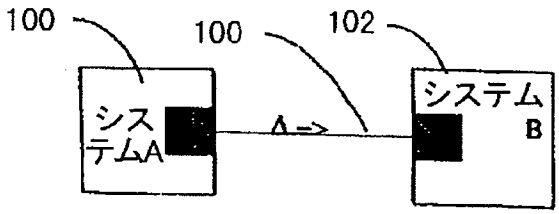
(21)出願番号　特願2001－287834(P2001－287834)
(22)出願日　平成13年８月17日(2001.8.17)
(31)優先権主張番号　０９／６４１０２８
(32)優先日　平成12年８月17日(2000.8.17)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　500160044
フュージョンワン・インコーポレイテッド
アメリカ合衆国、95113　カリフォルニア州、サン・ホーゼイ、アルマデン・ブールバード、55、スウィート・800
(72)発明者　デイヴィッド　エル　ミュルター
アメリカ合衆国　カリフォルニア州　95060　サンタ　クルーズ　イーストリッジ　ドライヴ　32
(74)代理人　100059959
弁理士　中村　稔　（外９名）

最終頁に続く

(54)【発明の名称】　データ転送および同期システム用のベースローリングエンジン

(57)【要約】
【課題】　様々なタイプの装置の間で個人情報を同期させること。
【解決手段】　第１データパッケージは、識別番号、アクション、複数のフィールドを有する。第２データパッケージは第１トランザクションの後に続く第２トランザクションを有する。第２トランザクションは識別番号、アクション、属性を持つフィールドを有する。ベースローリングエンジンは、第２トランザクションの識別番号が第１トランザクションの識別番号に対応するか決定し、第２トランザクションの識別番号が第１トランザクションのフィールドの１つと対応するか決定する。第１および第２トランザクションの識別番号が対応し、第２トランザクションのフィールドが第１トランザクションのフィールドの１つに対応する際、第１および第２データパッケージが結合される。結合データパッケージは第２データパッケージに置き換わり、第１データパッケージは消去される。



1

【特許請求の範囲】

【請求項１】　データ転送および同期システムに格納されるデータパッケージを崩壊させる方法であって、

識別番号と、アクションと、それぞれが変更情報を示す属性を持った、複数のフィールドと、を含む第１トランザクションを有する第１データパッケージを供給する工程と、

前記第１トランザクションの後に続くようにされ、識別番号と、アクションと、属性を持ったフィールドと、を含む第２トランザクションを有する第２データパッケージを供給する工程と、

前記第２トランザクションの前記識別番号が前記第１トランザクションの前記識別番号に対応するかどうかを決定する工程と、

前記第２トランザクションの前記フィールドが前記第１トランザクションの前記フィールドの１つに対応するかどうかを決定する工程と、

前記第１トランザクションおよび前記第２トランザクションの前記識別番号が互いに対応し、かつ、前記第２トランザクションの前記フィールドが前記第１トランザクションの前記フィールドの１つに対応する際に、前記識別番号を持つ結合トランザクションを有する結合データパッケージを定めるために、前記第１データパッケージと前記第２データパッケージとを結合する工程と、

前記第２データパッケージを前記結合データパッケージに置き換える工程と、を具備することを特徴とする方法。

【請求項２】　前記第１データパッケージを消去する工程を具備することを特徴とする請求項１に記載の方法。

【請求項３】　前記第１データパッケージと前記第２データパッケージとを結合する工程は、

前記第２トランザクションのアクションの前記タイプを決定する工程と、

前記第２トランザクションの前記アクションが「追加」である際に、「追加」アクションと、対応するフィールドと、前記第２トランザクションの前記属性と、を含む前記結合トランザクションを定める工程と、

前記第２トランザクションの前記アクションが「変更」である際に、「変更」アクションと、対応するフィールドと、前記第２トランザクションの前記属性と、を含む前記結合トランザクションを定める工程と、

前記第２トランザクションの前記アクションが「消去」である際に、「消去」アクションと、対応するフィールドとを含む前記結合トランザクションを定める工程と、を具備することを特徴とする請求項１に記載のデータパッケージを崩壊させる方法。

【請求項４】　データ転送および同期システムに格納されたデータパッケージを崩壊させる方法であって、

それぞれが、識別番号と、アクションと、それぞれが変更情報を示す属性を有する複数のフィールドと、含む複

2

数の第１トランザクションを有する第１データパッケージを供給する工程と、

前記第１トランザクションの後に続くようにされ、識別番号と、アクションと、属性を持ったフィールドと、を含む第２トランザクションを有する第２データパッケージを供給する工程と、

前記第２トランザクションの前記識別番号が前記第１トランザクションの前記識別番号の１つに対応するかどうかを決定する工程と、

前記第２トランザクションの前記識別番号が前記第１トランザクションの前記識別番号に対応する際に、前記１つの第１トランザクションを認識する工程と、

前記第２トランザクションの前記フィールドが前記識別された第１トランザクションの前記フィールドの１つに対応するかどうかを決定する工程と、

前記第２トランザクションおよび前記識別された第１トランザクションの識別番号が互いに対応し、かつ、前記第２トランザクションの前記フィールドが前記識別された第１トランザクションの前記フィールドの１つに対応する際に、前記識別番号を持った結合トランザクションを有する結合データパッケージを定めるために、前記第１データパッケージと前記第２データパッケージとを結合する工程と、

前記第２データパッケージを前記結合データパッケージに置き換える工程と、を具備することを特徴とする方法。

【請求項５】　データ転送および同期システムに格納されたデータパッケージを崩壊させる方法であって、

第１装置に関連する第１ベースデータパッケージを定めるために、第１複数のデータパッケージを崩壊させる工程であって、各データパッケージは、トランザクションと、前記第１装置のデータに適用されているすべての前記トランザクションとを有する工程と、

第２装置に関連する第２ベースデータパッケージを定めるために、第２複数のデータパッケージを崩壊させる工程であって、各データパッケージは、トランザクションと、前記第２装置のデータに適用されているすべての前記トランザクションとを有する工程と、

第３装置に関連する第３ベースデータパッケージを定めるために、第３複数のデータパッケージを崩壊させる工程であって、各データパッケージは、トランザクションと、前記第３装置のデータに適用されているすべての前記トランザクションとを有する工程と、を具備する方法。

【請求項６】　データネットワークに接続されたサーバにおける装置エンジン内に配置され、データ転送および同期システムに格納されたデータパッケージを崩壊させるベースローリング装置であって、

識別番号と、アクションと、それぞれが変更情報を示す属性を持った複数のフィールドとを含む第１トランザク

ションを有する第１データパッケージを供給する第１供給部と、

前記第１トランザクションの後に続くようにされ、識別番号と、アクションと、属性を持ったフィールドとを含む第２トランザクションを有する第２データパッケージを供給する第２供給部と、

前記第１トランザクションの前記識別番号が前記第２トランザクションの前記識別番号に対応するかどうかを決定する第１決定部と、

前記第２トランザクションの前記フィールドが前記第１トランザクションの前記フィールドの１つに対応するかどうかを決定する第２決定部と、

前記第１トランザクションおよび前記第２トランザクションの前記識別番号が互いに対応し、かつ、前記第２トランザクションの前記フィールドが前記第１トランザクションの前記フィールドの１つに対応する際に、前記識別番号を持った結合トランザクションを有する結合データパッケージを定めるために、前記第１データパッケージと前記第２データパッケージとを結合する第３結合部と、

前記第２データパッケージを前記結合データパッケージに置き換える置換部と、を具備することを特徴とする装置。

【発明の詳細な説明】

【０００１】（関連出願への言及）本出願は、発明の名称がすべて「データ転送および同期システム」である以下に示す出願、すなわち、２０００年１月２５日出願の米国特許出願第０９／４９０，５５０号、ならびに、２０００年１月２６日出願の米国特許出願第０９／４９１，６７５号および米国特許出願第０９／４９１，６９４号についての一部継続出願である。

【０００２】（著作権の告示）本特許文書の一部には、著作権保護を受ける題材が含まれている。著作権保持者は、何人が米国特許商標庁の包袋または記録に現れているような本特許文書すなわち本特許明細書をそのまま複写することに対して意義を唱えないが、その代わりこれらについてすべての著作権を保有する。

【０００３】（技術分野）本発明は、データが各システムで保持される形体とは無関係の２つのシステム間におけるデータ転送に関し、特に、システム間および装置間においてデータ通信を行う効果的な手段を提供することに関する。

【０００４】（背景技術）パーソナルコンピュータまたはワークステーションに限らず、計算に関連した装置が進歩してきている。パーソナル計算装置のタイプおよびフォーマットの両方が実質的に増えてきている。小型ハンドヘルド計算装置は、連絡情報、個人的情報、文書情報およびその他の情報を多数保持し、また、ユーザがファクシミリ通信、電子メールの送信およびその他の方法によるワイヤレス通信を行うことができるよう十分に高

性能なものとなっている。発展した携帯電話であっても、連絡情報を記憶し、ウェブをサーフィンし、テキストメッセージを供給するのに十分なメモリおよび処理能力を有している。これらの装置がますます高性能となっていることに加えて、これらの装置の間で情報を転送する必要性も同様に高くなってきている。

【０００５】様々な装置のタイプが多数売り出されており、それらの様々な装置の間で情報を同期させることがますます課題となっている。例えば、ある人物が、特定の個人情報管理アプリケーションを用いて、オフィスのパーソナルコンピュータに日程情報を記憶させているとする。この人物は、通常、それと同じ情報を、携帯電話、ハンドヘルドオーガナイザ（hand-held organizer）および多分家庭用パーソナルコンピュータで利用したいと思うであろう。そのうえ、その人物は、プレゼンテーションまたは作業文書のような、オフィスのコンピュータとの間で同期するファイルデータを必要とするノートブックコンピュータを持っているかもしれない。

【０００６】今までは、文書間および個人情報マネージャ間の両方で同期をとることは、装置間を直接接続することにより、一般的には、ある装置における個人情報マネージャと別の装置における個人情報マネージャのようなアプリケーションの間を直接接続することにより、または、中間同期マッピングプログラム（intermediarysync-mapping program）を用いることにより、行われてきた。

【０００７】このような同期の一例としては、3Com Palm（登録商標）シリーズ計算装置のような3Com Palm（登録商標）ＯＳベースのオーガナイザを一般的に用いるものがある。このオーガナイザは、独自の日程システムを用いるが、Symantec社ACT!、Microsoft社Outlook（登録商標）およびその他のシステムのような様々な異なる個人情報管理ソフトウェアパッケージを用いて、このオーガナイザ内のデータをユーザに同期させている。この例では、Puma Technology社Intellisync（登録商標）のような中間同期プログラムが必要とされる。Intellisync（登録商標）は、ハンドヘルド装置と、同型でないデータレコード間における情報データシステムおよびマップデータシステムを記憶するコンピュータと、の両方で動作するアプリケーションプログラムである。別の例では、Microsoft社Outlook（登録商標）のコンピュータベースのクライアントとMicrosoft社Windows（登録商標）CE "Pocket Outlook"アプリケーションとの間の転送のようなアプリケーション間の直接転送が可能である。それにもかかわらず、上記両方の場合では、同期をとることは、パーソナルコンピュータとパーソナル計算装置との間を直接接続することにより行われる。このような接続としては、一般的には、例えばクレードル（cradle）におけるPalm（登録商標）をパーソナルコンピュータケーブルにケーブルを介して直接接続するものがあり、こ

の接続はワイヤレスでも同様に行われる。

【０００８】これらの同期システムの１つの要素は、同期処理が、特定のデータベースが変更される間の時間を表現できなければならず、かつ、変更されたフィールドを置換するかどうかについて決定しなくてはならない、ということである。通常、これは、ある１つのデータベースが変更されたことと、第２のデータベースが変更されていないこととにより、測定される。両方のデータベースが同期の間において変更される場合もいくらかあろう。この場合には、同期動作を行う際には、なされた２つの変化のうちいずれを「勝ち（win）」とするのかを決定し、同期をとる間、他方の変化を置換しなくてはならない。一般的には、競合（conflict）が存在するかどうかを決定する決定物（determinant）によって、この競合をユーザに解決させるいくつかの手段が与えられる。

【０００９】技術的には、この方法による同期は、一般的には、システム間の全レコードをコピーすることにより達成される。あるレベルでは、ユーザは、一般的には、データフィールドを１つのアプリケーションから別のアプリケーションにマッピングし、かつ、どのデータフィールドを異なる装置の対応するどこのフィールドに割り当てるかを指定する必要がある。開発者がアプリケーションの様々なプラットフォームをしっかりとサポートする場合には、マッピングはほとんど必要とされない。

【００１０】多くの例では、同期の対象となるデータは、一般的には、アドレスレコード、連絡情報、日程情報、メモ、および、その他のタイプの連絡情報のようなテキストデータの形をとっている。ある例では、同期の対象となるデータは、バイナリフォーマットの実行可能ファイルまたはワードプロセッサに特化した文書である。文書の同期が必要とされる多くの場合には、同期ルーチンによって、単に、対象となっている文書が変更されたか否かを決定し、２つのファイルのうちいずれが新しいかを決定するために、時間をベースにした表現を用い、上記２つのファイルのうち古い方のファイルが実際に変更されていなければ、同期を達成するために古い方のファイルを新しい方のファイルに置換している。これは、Microsoft社のWindows（登録商標）に基づいたシステムにおけるよく知られた「ブリーフケース」で用いられているモデルである。両方のファイルが変更されている場合には、ユーザは、競合を解決する選択権を同期ルーチンによって与えられる。一般的に、このような同期方法は、同期リンクを介して転送すべき文書またはバイナリファイルの全帯域を必要とするので、あまり効率的なものではない。加えて、あるレベルでは、同期プログラムは、異なるプログラム間に所定のフィールドをマッピングするために、ユーザによる対話を必要とする。

【００１１】異なる計算装置間で同期をとることの難しさの１つとして、アプリケーション同士およびプラットフォーム同士が互いにいくらか異なっているということがある。これにもかかわらず、すべての同期プログラムは、通常、広範囲に利用することができるようにするために、所定の機能を必要とする。具体的には、同期プログラムは、様々なプラットフォームにおいて普及したアプリケーションを処理しなくてはならない。発生した同期間において、異なる装置間における同一の情報が変更される際には、同期アプリケーションは競合を解決しなくてはならない。同期プログラムは、連絡情報、電子メール、日程情報、メモもしくはその他の文書の形をとるテキストデータであろうと、または、特定タイプのフォーマットにより形成された文書またはプログラムの形をとるバイナリデータであろうと、あらゆるタイプのフォーマットにより形成されたデータについて、同期を行わなくてはならない。

【００１２】もっと広い意味では、全くタイプの異なる装置間でデータを効果的に同期させるアプリケーションによれば、例えばPalm（登録商標）計算装置のような個人情報マネージャハードウェア装置と、パーソナルコンピュータとの間で独自の個人情報を同期させること以上に、利益が得られる。個人情報管理（ＰＩＭ）装置とデスクトップシステムとの間においてデータ転送を確立する際にも顕著となる同じ目的は、異なるプラットフォーム上の他タイプの装置の間でデータ転送を行わなければならないアプリケーションの役に立つ。これらの目的には、スピード、低バンド幅、精度およびプラットフォームの独立性が含まれる。

【００１３】例えば、現在の電子メールシステムは、全体のメッセージまたはファイルが異なるシステム間でそのまま転送されるような、互いに全く異なる装置用の同期方法にいくらか類似したシステムを用いている。ユーザが電子メールに応答する際には、通常、オリジナルメッセージの全テキストが送信者に戻される。このとき、この送信者は、自分がもともと送信した電子メールテキストについては２コピー持っている。電子メールの添付ファイル（attachment）が変更され戻されたとしても同じようなことが起こる。両システム間で同一である全テキストは、本来、送信者のシステムでコピーが取られている。

【００１４】（発明の開示）本発明は、データ転送および同期システムに格納されたデータパッケージを崩壊させるベースローリングエンジンに関する。第１データパッケージが供給される。この第１データパッケージは、識別番号と、アクションと、複数のフィールドとを有する。各フィールドは、変更情報を示す属性を有する。また、第２データパッケージも供給される。この第２データパッケージは、上記第１トランザクションの後に続くようにされた第２トランザクションを有する。この第２トランザクションは、識別番号と、アクションと、属性

を持ったフィールドとを有する。ベースローリングエンジンは、第２トランザクションの識別番号が第１トランザクションの識別番号に対応するかどうかを決定する。ベースローリングエンジンもまた、第２トランザクションのフィールドが第１トランザクションのフィールドの１つに対応するかどうかを決定する。第１トランザクションおよび第２トランザクションの識別番号が互いに対応し、かつ、第２トランザクションのフィールドが第１トランザクションのフィールドの１つに対応する際に、第１データパッケージと第２データパッケージとが結合される。この結果、結合されたデータパッケージ（結合データパッケージ）は、上記識別番号を持った結合されたトランザクション（結合トランザクション）を有するように定められる。第２データパッケージは、結合データパッケージに置き換えられる。

【００１５】（発明を実施するための最良の形態）本発明の様々な実施形態を参照して本発明を説明する。本明細書および添付図面を参照すれば、本発明の他の特徴および利点が明白となろう。

【００１６】図１は、本発明の一実施形態に基づいて構成された第１データ転送および同期システムを示す一般化されたブロック図である。第１システムまたは装置であるシステムＡ、および、第２システムまたは装置であるシステムＢは、通信ライン１１０により接続されている。通信ライン１１０が、２つのシステムを直接接続するものであり、データがシステム間を通過できるようにしていることは、すぐに理解できよう。例えば、様々な実施形態として、このような接続には、シリアルポート接続、パラレルポート接続、イーサネット（登録商標）接続、他タイプのネットワーク、赤外線接続、およびこれらと同様なものが含まれる。様々な実施形態として、システムＡおよびまたはシステムＢは、パーソナルコンピュータ（ＰＣ）、スマート電話、携帯電話、個人情報計算装置、ハンドヘルドコンピュータ、ノートパソコンおよびウェブブラウザである。その他の実施形態として、システムＡおよびまたはシステムＢは、コンピュータシステムにおけるハードウェア構成要素、ならびに、例えば受信しかつ他の装置に対して情報を提供するように設けられたプロセッサおよびメモリを含むハードウェアを別の方法で組み合わせたものを含む。その他の実施形態としてのシステムＡおよびまたはシステムＢは、そのような情報を含みかつハードウェアの集合に存在するソフトウェアを含む。そのようなソフトウェアの一例には、個人情報マネージャのようなアプリケーションが含まれる。この個人情報マネージャは、Microsoft社のWindows（登録商標） NT オペレーティングシステム、Unix（登録商標）オペレーティングシステム、Linuxオペレーティングシステム、および、異なるタイプのアプリケーションフォーマットに翻訳される、バイナリフォーマットを有するファイルタイプを格納することが可能なその他のシステムにより用いられるような、連絡情報およびその他のそういった情報、電子メールシステムならびにファイルシステムを含む。

【００１７】図１において、システムＡは、相違送信機を表す機能ブロック１００を含む。システムＢは、相違受信機を表す機能ブロック１０２を含む。相違送信機１００は、この送信機を動作させる制御信号を受信することにより、システムＢに送信される情報の指定されたデータ構造を調べる。相違送信機１００は、このような情報をシステムＡから抽出し、抽出した情報を相違情報Δに変換する。相違情報Δは、システムＢで発生したシステムＢに対する変更事項、および、これらの変更事項を実行するための指示のみを含む。これにより、転送すべきデータがシステムＢに存在するファイルに対する変更事項であれば、相違情報Δは、そのようなファイルにおける相違点およびそのような相違点が発生する場所のみを含む。システムＢにデータが全く存在しなければ、相違情報Δは全ファイルとなる。システムＢにおける相違受信機１０２が受信した相違情報Δは、システムＢにおいて再構築され、相違情報Δ内に反映された変更事項は、システムＢにおいて更新される。例えば、システムＡおよびシステムＢがともにコンピュータであり、かつ、システムＡの所定バイナリファイルを更新する必要があるならば、システムＡにおける相違送信機は、システムＢに存在する既知ファイルおよび新ファイルにおける相違点を抽出し、それらの相違点（これらの相違点を挿入する場所についての指示）のみを相違受信機１０２に送信する。相違受信機１０２は、相違情報（Δ）を解釈し、システムＢにおけるバイナリファイルを再構築する。この方法によれば、システム間で全バイナリファイルを転送しなくとも、システムＢにおける情報は更新される。

【００１８】図２は、本発明の一実施形態に基づいて構成された第２データ転送および同期システムの一般化されたブロック図を示す。図２において、システムＡおよびシステムＢは、それぞれ相違同期装置を表す機能ブロック１０４を含む。同期装置１０４の機能は、送信機と受信機とを組み合わせたものの機能と同様である。すなわち、同期装置は、相違情報Δを送信および受信の両方を行うことができる。例えば、システムＡおよびシステムＢは、それぞれポータブルコンピュータおよびデスクトップコンピュータである。連絡情報のような情報が２つのシステム間で同期する必要がある際には、両システムが同一データを確実に保持することができるようにするために、相違同期装置１０４は、システムＡおよびシステムＢのいずれかにおける連絡情報になされた変更事項を抽出し、上記システム間で情報Δを所定回数だけ送信し、送信側システムからの情報を更新するために受信側システムにおけるデータを再構築する。

【００１９】図３は、本発明の一実施形態に基づいて構

成された第３データ転送および同期システムの一般化されたブロック図を示す。繰り返すが、システムＡは相違送信機を含み、システムＢは相違受信機１０２を含む。この実施形態では、格納サーバ３００がシステムＡとシステムＢとの間に接続されている。格納サーバ３００は、システムＡにより供給される相違情報Δの独立したデータベースを格納する。このデータベースによって、システムＡは、第１の時点において格納サーバ３００に対して相違情報Δを供給することができ、格納サーバ３００は、第１の時点とは同一でない第２の時点において、上記と同一の相違情報Δをシステムに供給することができる。加えて、相違情報Δの多重セットは、それぞれ異なる時点において供給され、後の検索のためにシステムＢにより格納される。さらに、相違情報のセットは、システムＡまたはシステムＢのいずれかにおけるデータを以前の状態に戻すことができるように、サーバ３００に保持される。

【００２０】再度繰り返すが、格納サーバ３００は、システムＡおよびシステムＢの両方に対して直接接続１１０により接続されている。格納サーバ３００は、具体的には、受信機１００からの相違情報Δを受信しかつこの相違情報Δを送信機１０２に対して供給するように設けられたサーバである。一実施形態として、サーバ３００は、このような転送を実現するための特定の機能ルーチンを含む。これに代えて、サーバ３００は、ファイル転送プロトコル（ＦＴＰ）またはハイパーテキスト転送プロトコル（ＨＴＴＰ）のような標準インターネット通信プロトコルに応答する、標準情報サーバタイプを具備する。

【００２１】図４は、本発明のシステムのさらに別の一実施形態を示し、この実施形態では、システムＡおよびシステムＢは、直接接続ライン１１０により格納サーバ３００に対して再度直接接続されており、システムＡおよびシステムＢのそれぞれが、相違同期装置１０４を含む。相違情報Δは、第１の時点において、同期装置１０４を介してシステムＡに送られかつシステムＡから送られ、格納サーバ３００に送られかつ格納サーバ３００から送られ、第２の時点において、システムＢに送られかつシステムＢから送られる。この実施形態では、格納サーバ３００は、各システムが同期した最新時点の後、システムＡおよびシステムＢの両方で別々に変化したデータの間での競合を解決するための、後述するようなルーチンを含む。

【００２２】図５は、４つのシステム、すなわち、相違同期装置１０４を含むシステムＡ、相違受信機１０２を含むシステムＢ、相違同期装置１０４を含むシステムＣおよび相違送信機１００を含むシステムＤを含む、本発明のさらに別の一実施形態を示す。各システムが格納サーバ３００に直接接続されているので、異なるシステムの間での相違データΔの転送を制御することが可能である。サーバ３００は、システムＡからシステムＤを具備しかつ相違情報Δにおける様々な構成要素を様々なシステムのそれぞれに対して送信することを制御する、様々なタイプのシステムを調査するルーチン（これの詳細については後述する）を含む。例えば、システムＢは相違受信機１０２だけしかを含まないので、システムＢに供給される相違情報Δ_2は、格納サーバ３００とシステムＡとの間で転送される相違情報の副構成要素でもありうるし、または、単にシステムＤからの受信同報情報Δ_4でもありうる。本発明のシステムの一実施形態では、サーバ３００自体は、各受信機／送信機／同期装置から得られた相違情報の経路を定めない。サーバ３００は、上記情報のレポジトリ（repository）として動作し、どの相違情報Δがどの受信機／送信機／同期装置のものであるかについては、各受信機／送信機／同期装置により決定される。

【００２３】図６は、同期装置が格納サーバ３００に設けられている、さらに別の一実施形態を示す。回送機（forwarder）およびまたは受信機を同様にサーバ３００に設けることも可能であることに認識されたい。ここで示す特定の実施形態は、装置が携帯電話である場合のように、装置の処理パワーおよびメモリが限定されるような場合に有利となろう。このような実施形態においてシステムＡと装置エンジン１０４ａとの間で転送されるデータが相違情報であるか否かが、システムＡが相違情報を検出し出力する能力を有しているかどうかに依存するであろう、ということに注意されたい。各装置は、相違受信機、相違送信機または相違同期装置を含む。相違同期装置の一部がシステムＡに存在し、他の部分がサーバ３００に存在しうることを理解されたい。

【００２４】図７は、図６に示すような複数の装置が、インターネットのような公衆または私用ネットワーク７００を組み合わせたものに接続される、本発明のさらに別の実施形態を示している。ネットワーク７００は、１つ以上の格納サーバ３００_1、３００_2を含み、この場合には、このようなサーバのうち１つにおける中間記憶領域を介して各装置の間で送信される相違情報Δを含む。ネットワーク７００は、相違回送装置、受信機または同期装置を含むように具体的に設計されたサーバのような１つ以上の特殊な機能サーバに対して、装置を接続する。図７における装置は、限定しないとすると説明上、オフィス用パーソナルコンピュータ（ＰＣ）７０２と、スマート電話または携帯電話７０４と、個人情報Palm（登録商標）計算装置７０８と、家庭用ＰＣ７１０と、ウェブブラウザ７１２とを具備する。格納サーバ３００_1、３００_2に存在するデータが、特定の受信機または同期装置がこれらが設けられた装置におけるデータを同期させるために用いる相違情報を含むかどうかを決定するために、装置７０２－７１２に存在する各相違受信機、各相違送信機およびまたは各相違同期装置は、格納サー

バ３００$_1$、３００$_2$に格納されたデータをポーリングする手段を含む。

【００２５】以下の記載では、相違受信機、送信機および同期装置について記載する実施形態を、データの同期をとることに関連して、複数の異なる装置の間で、連絡情報、日程情報およびバイナリファイル情報を同期させる際に使用する場合を参照し、説明する。本発明のシステムが、同期アプリケーション、または、連絡情報もしくは日程情報のような特定タイプのデータに依存するアプリケーションに、限定されないことは、容易に理解されよう。具体的には、異なるシステムおいてデータを抽出しかつデータを再構築するルーチンにより、２つのシステムの間におけるデータの変更事項のみを具備するデータを送信することは、効率的なデータ伝送の進歩に資することができる。本発明によれば、データに対する変更事項のみが伝送されるので、２つのシステムの間でデータを転送するために用いるバンド幅を効果的に削減できることが考慮されている。この結果、伝送すべきデータは、システム間で全ファイルが転送されるとしたら転送されるであろうデータよりも実質的に小さくなるので、このようなトランザクションが発生するスピードを速くすることができる。

【００２６】一般的には、システムは、相違送信機１００、相違受信機１０２および装置エンジンの形をとる相違同期装置１０４のそれぞれの機能を提供するクライアントソフトウェアを具備する。この装置エンジンは、この装置エンジンが起動する装置のタイプに固有の構成要素を少なくとも１つ含んでおり、この構成要素によって、装置から情報を抽出し、この情報を相違情報に変換し、かつ、格納サーバに対してこの相違情報を伝送することが可能となる。これにより、本発明のシステムに接続されたあらゆるシステムを通して、情報の応答を行うことができる。本発明のシステムで利用される格納サーバ３００は、インターネットサーバまたはＦＴＰサーバのような、どのようなタイプの格納サーバであってもよいし、インターネットサービスプロバイダ（ＩＳＰ）のようなどのようなソースからも供給されうるけれども、有効であって、かつ、本発明の部品として接続されるシステムの間で情報を最適に転送できるようにカスタマイズされる格納サーバのの独特な面について、以下に説明する。装置間でインターネット接続を利用できる限りにおいて、本発明の同期システムを用いてこれらの装置を同期させることができる。装置間または装置とサーバとの間で、同一時点にインターネット接続を形成しておく必要はなく、かつ、いかなる時点においても情報を損失することなく本発明のシステムに新しい装置を追加することができる。このシステムは、情報に対して全体的に明白なアクセスを行い、かつ、各装置における装置エンジンは、本発明にしたがって、個人情報サービスをシームレスに統合することが可能となるようにオペレーティングシステムを自由に拡張することができる。加えて、本発明のシステムにおける他のシステムに転送する必要がある、情報に対する変更事項は、非常に早い応答時間を可能とするように伝送される。本発明のさらに別の面として、この方法で転送される情報は、インターネットの公衆部分での安全性を確保するために、暗号化される。

【００２７】図８は、本発明の一実施形態に基づいて構成されたデータ転送および同期システムのシステム構造を示す一般化されたブロック図である。この実施形態では、本発明のシステムによれば、ある人物が個人情報を取り扱う際に用いる、パーソナル装置およびアプリケーションの集合を接続することができる。このシステムは、やはり様々な装置タイプに対して公衆または私用情報を放送するために使用されうる。本発明のシステムの一部として示される、各装置についての装置エンジンの形をとるシステムソフトウェアは、同期を可能とするために、装置の集合を通して配布される。装置エンジンは、例えばインターネット接続を介して転送されるインストールパッケージにより配布される。本質的には、本発明の装置エンジンソフトウェアは、システムにおけるすべての情報を完全に同期させることを維持する分散処理ネットワークを形成する。このサービスの配達に伴う処理負荷は、大きなアプリケーションに対してシステムを容易に評価させる末端装置に押し付けられる。

【００２８】図８には２つのタイプの装置エンジンが示されている。一方のタイプは、様々な装置に配置されており、変更データをサーバに対して出力し、もう一方のタイプは、サーバに組み込まれており、装置に生成された変更情報をデバイスから受信する。これに代わる一実施形態には、２つの装置エンジンのハイブリッドが含まれる。すなわち、装置エンジンの一部が、装置とサーバにおける一部とに存在している。

【００２９】図８に示すように、任意数および任意タイプの装置８０２～８０８が、本発明のシステムに従って用いられうる。電話８０２は、携帯電話または標準ＰＯＴＳ接続電話を含む。電話８０２は、連絡情報、新世代携帯電話にサポートされているように、データ構造８１２に格納された予約およびタスクデータを含む。このような情報を含むアプリケーションデータ８２２を利用するアプリケーション８１２は、すべて電話ユニット８０２に格納される。同様に、Palm（登録商標）計算装置８０４のような携帯情報端末は、アプリケーション８１４と、連絡、予約およびタスクのような情報を含み、さらに、ＰＤＡ８０４により生成され格納される文書のようなファイル情報をも含むアプリケーションデータ８２４とを含む。装置８０６は、Microsoft社のWindows（登録商標） 95,98,NTまたは2000のようなオペレーティングシステムを起動するWindows（登録商標）パーソナルコンピュータとして表現される。装置８０６上で起動する

13

アプリケーション８１６は、Windows（登録商標）オペレーティングシステム自体、Microsoft社のOutlook、Symatec社のACT 個人情報マネージャ、Goldmine Software社のGoldmine、Lotus社のOrganizer、Microsoft社のインターネットエクスプローラウェブブラウザ、Netscape社のCommunicator Suite、Qualcomm社のEudora e-mail、およびその他の様々なプログラムを含み、これらのそれぞれは、システム８０６の外部にある装置と同期をとるのみならず、システム内にある装置間およびアプリケーション間において同期をとる必要があるアプリケーションデータ８２６の独自セットを有している。最後に、アプリケーションデータ８２８の独自セットを有するウェブポータルアプリケーション８１６にインターネットを介して接続している、専用ウェブブラウザクライアント８０８が示されている。独自のハードウェアに実質的にアプリケーションおよびアプリケーションデータを格納する装置８０６とは違って、ウェブポータルアプリケーションは、別々のサーバに提供されているとともに、インターネット接続を介してブラウザ８０８にも提供されている。やはり、ポータルアプリケーションプロバイダに格納されているウェブポータルアプリケーションは、ユーザが同期をさせたいと思うアプリケーションデータ８２８のセットを含む。例えば、Yahoo!およびSnap.comのような大きなウェブポータルは、それらのユーザに対して、free e-mailおよび連絡情報の保存のようなサービスを提供している。ユーザは、これを自分の携帯電話、ＰＤＡまたはWindows（登録商標）装置上で起動するアプリケーションと同期させたいと思うであろう。

【００３０】図８に示す各システムの特定アプリケーションデータにアクセスするために、装置エンジンは各タイプの装置と関連している。携帯装置エンジン８６２は、携帯電話のアプリケーションデータ８２２と通信し、かつ、アプリケーションデータ８２２に組み込まれている。同様に、ＰＤＡ装置エンジン８６４が同様に設けられている。このＰＤＡ装置エンジン８６４は、Palm（登録商標）オペレーティングシステム、Windows（登録商標）CE オペレーティングシステム、または、その他のＰＤＡ型オペレーティングシステムのいずれかを必ずベースにしている。Windows（登録商標）をベースとした装置エンジン８６６は、サポートされたWindows（登録商標）アプリケーション８１６からアプリケーションデータ８２６を抽出するといった、後述するメカニズムを含んでおり、ウェブサービス装置エンジン８６８は、ウェブポータルアプリケーション８１８からアプリケーションデータ８２８を抽出するために合体している。

【００３１】図８に示すように、デバイス全体に供給されている装置エンジン（ここでは「デスクトップ装置エンジン」という。）もあれば、後部サーバ（格納サーバ

14

８５０または図９Ｂに示すような特定サーバを具備する後部サーバ）に構成要素を含んでいる装置エンジンもある。これは、一般的には図８における実線８３２、８３４、８３６および８３８により示される。また、図８には、点線８５５の上に位置する構成要素は、本発明のシステムの管理者またはサービスプロバイダにより提供される。装置エンジン８６２、８６４、８６６および８６８のそれぞれは、これらが設けられた装置のタイプに関連して構成される。例えば、携帯電話装置エンジン８６２はこの携帯電話に設けられる１つ以上の構成要素を含む一方、その他の構成要素はサーバ８５０に存在する。逆に、装置エンジン８６６は、その装置全体がWindows（登録商標）装置に存在する。

【００３２】各装置からのデータは、インターネット接続７１０を介して格納サーバ８５０に接続される。上述したように、格納サーバ８５０は、包括的な格納サーバ、または、後述するような本発明のシステムを用いて使用するために具体的に設けられた格納サーバであろう。１つ以上の格納サーバ８５０は、システム８０２、８０４、８０６および８０８の集合の間でトランザクションを通信するために用いられる。任意数の様々なタイプのシステム８０２、８０４、８０６および８０８が、本発明にしたがって設けられるとともにシステム内に組み込まれることは、容易に理解できる。しかしながら、簡略化のために、現在使用中または開発中である市場で入手可能な計算装置（本発明のシステムが組み込まれるであろう計算装置）の様々なタイプをすべて列挙しているわけではない。

【００３３】最もシンプルな実施形態では、格納サーバ８５０は、データ処理能力をもたない単なる格納サーバであるが、装置エンジンのそれぞれは、システムにおけるその他の装置エンジンによりアクセス可能な特定位置に格納されるように、この格納サーバ８５０に対して相違情報のみを送信する。一実施形態では、各装置エンジンは、システム全体を完全に同期させた状態を維持するために必要な全処理を実行する。ある特定時点において、１つの装置エンジンだけしか格納サーバ８５０に接続する必要がない。これにより、切断方法を適用した多重システムを同期させることができる。各装置エンジンは、最新の同期がなされてから生じた変更事項を含むすべてのトランザクションをサーバからダウンロードし、これらの変更事項を特定の装置に対して適用する。

【００３４】変更事項または相違情報（Δ）は、１つ以上のデータパッケージに供給される。ここで、このデータパッケージの構造について説明する。各データパッケージは、アプリケーションデータ、ファイル、フォルダ、アプリケーション設定およびこれらと同様なものに限定されないものを含んだ、あらゆる装置にわたる任意および全転送情報に対する変更事項を記述する。各装置エンジンは、特定の装置エンジンに取り付けられた特定

ローカル装置８０２、８０４、８０６または８０８に適用される様々な階級の情報を含むデータパッケージをダウンロードすることを制御可能である。例えば、装置エンジン８６２は、アプリケーションデータ８２２における連絡情報および電話番号を記述する情報に対する変更情報を扱う必要があるのみであるが、装置エンジン８６６は、アプリケーションデータ８２６がアプリケーションデータ８２２よりも非常に大量となってからは、電子メールに対する変更事項、連絡情報および住所情報だけでなく文書ファイルおよびメモに対する変更事項をも扱う必要がある。

【００３５】各装置エンジンは、インターネット接続７１０を介して送信されたデータパッケーに対する暗号化および圧縮化を可能とする圧縮化／伸張化および暗号化／解読化構成要素を含む。データパッケージに対する圧縮化および符号化については、任意に設けられるということを認識されたい。各装置エンジンは、データパッケージを、アプリケーションデータ格納部８２２－８２８における情報タイプに要するローカルフォーマットに適用するために必要なマッピングおよび変換処理を実行する。また、装置エンジンは、最新の更新がなされて以来、ユーザが２つの異なるシステムにおける特定データフィールドに対するデータを同時に変更した場合に、不明瞭な更新を調査することを可能となる構成要素を含む。この場合には、装置エンジンは、ユーザがこのような事態に気付かせるメカニズム、および、ユーザが競合を解決できるようにするメカニズムを含む。

【００３６】図９Ａは、包括アプリケーション８１０および包括格納サーバ８５０とともに用いられる装置エンジンの一例を示す。具体的には、すべての処理はこの装置上で発生し、相違情報のみしかサーバ８５０に送信されないので、図９Ａにおける装置エンジンはデスクトップ装置エンジンである。やはり、デスクトップ装置エンジンを理解することは、サーバ側装置エンジンを理解することの手助けになる。このサーバ側装置エンジンについて以下に説明する。図９には、ブロック形式による装置エンジンの機能的構成要素、および、これらの構成要素間の相互関係が示されている。装置エンジン８６０は、図１－７に示した相違シーケンサ１０４の機能的ブロックと均等なものである。特定アプリケーションにより必要とされるような転送専用（相違送信機）または受信専用（相違受信機）の能力のために、必要な際に一部の機能性が用いられる。

【００３７】システムにおいて、装置についてのユーザの個人情報ネットワークを構成する各装置および全装置について、装置エンジンが存在する。図９Ａに示すように、各装置エンジン８６０は、アプリケーションオブジェクト９１０を含む。このアプリケーションオブジェクトは、各特定アプリケーション８１０に固有のものであり、かつ、装置エンジンと本発明のデータ転送システムのバランスとの間における標準インターフェイス、および、装置エンジンとアプリケーションデータ８１０との間の標準インターフェイスを提供する。以下、このアプリケーションオブジェクトの詳細について説明する。アプリケーションオブジェクトは、広範囲な種類のコンピュータメーカ独自のアプリケーションをサポートする接続可能な構成のことである。アプリケーションオブジェクトの役割は、アプリケーションデータに対してアクセスするために任意数の標準インターフェイスを介してアプリケーションに接続することにより、アプリケーションからのデータを一時的すなわち「汎用」データ構造にマッピングすることである。アプリケーションオブジェクトのデータ構造は、格納サーバに対して供給するデータパッケージを生成するために、装置エンジンの構成要素により用いられる包括的または「汎用なデータ」フォーマットにデータを変換することである。

【００３８】前回データの抽出および同期が行われた後のある時点における装置のデータのコピーを含む、アプリケーションオブジェクト格納部（ＡＯＳ）９２０もまた設けられている。アプリケーションオブジェクト格納部９２０は、装置エンジンにおけるアプリケーションオブジェクト９１０からのデータの前回の状態についてのスナップショットを格納する、反映型インターフェイスである。ＡＯＳのサイズは、各装置エンジンにより収集されているデータに依存する。

【００３９】アプリケーションオブジェクトの包括的な出力は、デルタモジュール９５０に供給される。デルタモジュール９５０は、アプリケーションオブジェクト９１０の出力と、アプリケーションオブジェクト格納部（ＡＯＳ）９２０に供給されるデータのコピーとの間における、データについての相違を計算する。実際の相違およびパッチルーチンは、ＸデルタまたはＹデルタのようなルーチンを具備する。デルタモジュール９５０は、本明細書のある部分ではこれに代えて「ＣＳ構造デルタ」と称される。加えて、相違情報は、これに代えて「変更ログ」と称される。各変更ログ（または相違情報のセット）は、自己記述（self describing）シリーズの同期トランザクションである。後述するように、変更ログは、ネットワークに出力される前に、暗号化および圧縮化される。

【００４０】これにより、同期の間においては、アプリケーションオブジェクトは、後述するメカニズムを用いることにより、装置における各アプリケーションからデータを抽出し、抽出したデータを汎用データフォーマットに変換する。この後、デルタモジュールは、アプリケーションオブジェクト出力とＡＯＳ出力とを比較することにより、相違セットを生成する。この相違情報は、データパッケージの形をとる格納サーバ８５０の出力のための暗号化および圧縮化ルーチンに転送される。これに代えて、１つのアプリケーションからのデータは、別の

17

アプリケーション、例えば図１０における矢印１０５０が示すWindows（登録商標）の環境におけるデータに同期させるために用いられうる。

【００４１】アプリケーションオブジェクトが、非構造的バイナリデータまたは構造的アプリケーションデータとの間の直接的なインターフェイスとなりうることは、明確に理解されよう。相違ルーチンは、比較生成において２つのデルタモジュール９５０を用いることをサポートしている。

【００４２】アプリケーションオブジェクトおよびデルタモジュールの動作は、ＰＤＡのアプリケーションのようないくつかのアプリケーションがそのデータへの変更事項を出力する能力を有するという事実によって簡略化される場合がいくらかある。このような場合には、ＡＯＳとの比較は必要でない、すなわち、アプリケーションは既にこのアプリケーションのデータになされた変更事項を調査するメカニズムを含んでいるので、デルタモジュール９５０は、データパッケージにデータを供給するだけでよい。しかしながら、多くの場合、アプリケーションは、Microsoft社のOBDCインターフェイス、Microsoft社の標準アプリケーションプログラミングインターフェイス（ＡＰＩ）またはその他の同様な標準インターフェイスのような、データにアクセスするための標準インターフェイスをせいぜい提供するのみである。

【００４３】さらに、装置エンジン８６０は、データパッケージにおけるオブジェクトごとにバージョン番号を付与するバージョンモジュールを含む。後述するように、データパッケージにおける各オブジェクトは、汎用的に固有ＩＤ（ＵＵＩＤ）が割り当てられる。これにより、以前の多くの同期システムとは異なり、本発明のシステムは、２セットのデータの時間スタンプを比較することのみでは、データを同期させない。バージョンモジュール９１５によって、各装置エンジンは、どのデータパッケージを適用するかを決定するために、格納サーバに供給されているデータパッケージに対する最新の同期の状態を調べることが可能となっている。これにより、装置エンジンは、別の装置エンジンが格納サーバに対する変更事項をアップロードする回数とは無関係に、この装置エンジン自体を同期させることができる。別言すれば、第１の装置エンジンは、第２の装置エンジンがサーバに対してデータパッケージを何回アップロードするかを問題としない。

【００４４】事象モジュール９２５は、同期初期化事象を制御する。いつ同期を行うか、どのように同期を行うかのようなアイテムは、デルタモジュール９５０が同期動作を実行するきっかけとなる。

【００４５】ユーザインターフェイス９３０は、装置エンジン８６０が接続されている特定装置のシステムユーザに対して、付加的な機能的特徴を割り当てるために設けられる。このユーザインターフェイスは、競合解決モジュール９４０、フィルタリングモジュール９４５およびフィールドマッピングモジュール９３５に接続されている。これらのモジュールのそれぞれは、全同期プログラムおよびユーザが期待するかの両方に必要な機能性を提供する。

18

【００４６】フィルタリングモジュール９４５は、例えばフィールドレベルコンテンツサーチに基づいた、様々なタイプのコンテンツ用のフィルタリングを行う。フィールドマッピングモジュール９３５によって、ユーザは、文書ストリームに供給されたアイテムの所定変換を再マッピングする。例えば、装置エンジン８６０がパーソナルコンピュータを作動させていれば、このパーソナルコンピュータとノートブックコンピュータとの間には同期が発生しており、ユーザは、このノートブックコンピュータ上の異なるディレクトリにマッピングしたいと思っている、パーソナルコンピュータ上に「マイドキュメント」ディレクトリを持っており、フィールドマッピングモジュール９３５は、発生する再マッピングを考慮する。フィールドマッピングモジュールはデータパッケージ出力を指示する際における変更事項を考慮するということを認識されたい。フィールドマッピングモジュール９３５は、フィールドマッピングおよび同期アプリケーションを使用するような従来方式とは異なり、例えば、Microsoft社のOutlookのような１つのアプリケーションからの連絡情報の特定データフィールドを、Symantec社のACTのような別のアプリケーションにマッピングする必要はない。

【００４７】さらに、デルタモジュール９５０は、圧縮化モジュール９７０および暗号化モジュール９６０に接続されている。圧縮化モジュールをイネーブルにする必要がないということを認識されたい。一般的なPK ZipまたはWinzipモジュールまたはHiFn Corporationより入手可能なモジュールのような、任意タイプの圧縮モジュール９７０を、本発明に従って利用することができる。さらに、MD5、RCH6、TwoFish、Blowfishまたはその他の対称暗号化アルゴリズムを用いることもできる。本発明の一実施形態では、圧縮化を行わない暗号化が用いられる。本発明の題２実施形態では、暗号化を行わない圧縮化が用いられる。本発明の第３実施形態では、圧縮化も暗号化も用いられず、本発明の第４実施形態では、圧縮化および暗号化の両方が用いられる。

【００４８】また、バージョンモジュール９１５によって、装置エンジン８６０は、独特な同期プロファイルを持った多数のユーザをサポートすることができる。これにより、同一のマシンにアクセスしている多数のユーザは、それぞれ、同一の装置エンジンを用いて、各自のデータセットを同期させることができる。例えば、特定装置におけるアプリケーション８１０がMicrosoft社のExchangeサーバに接続されたパーソナルコンピュータ上にMicrosoft社のOutlookを備え、かつ、Outlookがマルチユ

ーザのプロファイルを有するように構成されている場合には、バージョンモジュール９１５は、同期要求が発生する際に装置エンジンを通して適用されるデータを調査する。これにより、異なるデータセットにアクセスする、同一のOutlookクライアントソフトウェアの２つのユーザは、同一のマシンを介して、同一の装置エンジンおよび本発明のシステムを利用することができる。さらなる実施形態では、特定装置は、同一の接続を介してシステムにアクセスする全く異なる装置のユーザをサポートする。例えば、Palm（登録商標）装置は、コンピュータおよびまたはインターネット接続に接続するために、クレードル（cradle）を用いる。特定ユーザが、別ユーザのPalm（登録商標）pilotを同期させるために、別ユーザに自分のPalm（登録商標）pilotクレードル接続を使用してほしいのであれば、装置エンジンは、全く異なる装置のローカルアプリケーションオブジェクト格納部を更新するために、データパッケージを生成することができる。したがって、アプリケーションオブジェクト格納部は、全く異なる装置の同期を実現する場合の一時的な格納部として用いられうる。

【００４９】相違ｔエンジン９００の出力は、格納サーバ８５０に出力されるデータパッケージを備える。上述したように、所定時間には、１つの装置エンジンのみを格納サーバ８５０に接続すればよい。データパッケージは、別の装置エンジンにより格納サーバの特定一実施形態としてに対する要求がなされるまで、格納サーバ８５０に格納されうる。同様に、デルタエンジン９００は、本発明のシステム内の同期されたデータへアクセスするために、格納サーバにおける別の位置を問い合わせることができる。格納サーバの領域に対するアクセスは、以下にさらに具体的に説明するマネジメントサーバ（ＭＳ）により制御される。一実施形態では、各同期動作時には、装置を立証しかつ格納サーバにおける各装置のデータパッケージの位置を装置エンジンを供給するために、各装置についての装置エンジンがマネジメントサーバにログインすることが要求される。

【００５０】データパッケージは、ストリーミングフォーマットの格納サーバより装置エンジンに効果的に供給されるので、バンド幅の最小値および装置の格納部を用いて、処理を発生させることができる。装置エンジン８６０および特にデルタモジュール９５０は、バージョン情報およびアプリケーションオブジェクト格納部９２０に存在する反映型データに基づいて、データパッケージを変換する。格納サーバ８５０からデルタモジュール９５０に対してデータが戻される際には、デルタモジュールは、後にデルタ情報をアプリケーション８１０用に利用される特定インターフェイスに変換する特定アプリケーション用のアプリケーションオブジェクト９１０に対して、相違データを戻す。装置エンジンは、一旦入力ストリームからの全データパッケージを完全に適用する

と、ローカルシステムになされた変更事項を記述した一連のデータパッケージを生成する。装置エンジンは、最新同期済バージョンの各アプリケーションの実データを調査し続けるために、ローカルアプリケーションオブジェクト格納部９２０を用いる。このデータは、この後、次の同期要求時にデルタモジュールが次のデータを比較する際に用いられる。発生したデータパッケージは、上述したような不明瞭な事態を解決することに起因して生成された、動作および暗号化変更事項を含む。

【００５１】図９Ｂには、サーバをベースとした装置エンジンが本発明のシステムにどのような設けられるかが描かれている。この実施形態で示すPalm（登録商標）の例、ここでは、Palm（登録商標）装置が、インターネットおよびサービスプロバイダのデータセンタ９００に直接接続できる能力を有する例が示されている。このデータセンタは、データセンタ９００に存在するサーバとの間で未許可の通信が行われるのを防止し、かつ、データの完全性を保護するために、ファイアウォール９７５を含んでいる。格納サーバ８５０は、マネジメントサーバ（ＭＳ）１４１０がそうであるように、ファイアウォールを通して直接通信を行うことができる。この図には、それぞれが１つの特定タイプのアプリケーションを同期させる目的で設けられた、２つの同期サーバ９８２および９８４が示されている。同期サーバ９８２はPalm（登録商標）装置専用として設けられている一方、同期サーバ９８０は、例えばポータルアプリケーション（ポータル１）専用として設けられている。

【００５２】Palm（登録商標）装置８０４ａは、そのデータに対する変更事項を送信するメカニズムを含んでいるので、データは、ＡＯＳにおけるデータに対する相違計算および更新が実行された後に変更事項がPalm（登録商標）８０４ａにダウンロードされうるような同期サーバ９８２に対して、ファイアウォール９７５を介したＨＴＴＰ要求および応答を用いて、送信されうる。

【００５３】アプリケーションにおける同期サーバは、ユーザのデータに対して同時に発生した同期を扱う。各同期サーバは、実行可能な同一の同期サーバを用いて、同期対象となる多数の装置に対するプラグインサポートを含む。各装置タイプは、同期時にどのＡＯ／ＡＯＳ構成要素を用いるのかを識別する独自の名前を有する。

【００５４】同期サーバは、この同期サーバ内部の同期相違エンジン、ＡＯＳおよびＡＯのような外部エンティティに対してデータを送信する際、ならびに、この外部エンティティから検索をする際に、汎用データレコードの概念を用いる。よって、Palm（登録商標）アプリケーションでは、サーバＡＯの役割は、単に、装置特有のフォーマットのレコードを獲得して汎用レコードフォーマットに変換することである。

【００５５】同期サーバは、サードパーティのアプリケーションパートナーが自分のサービスをサーバ内に容易

21

に追加できるように、プラグイン構造を有する。現在では、サーバがMicrosoft社のWindows（登録商標） NTサーバ内で動作していれば、同期サーバは、Windows（登録商標） NT レジストリを介して同期構成要素を発見する。別の実施形態では、このような機能は、サーバにおけるＡＯおよびＡＯＳのそれぞれによる処理を維持するために、各同期サーバ上で動作するコンポーネントマネージャ（Component Manager）で実行される。ＡＯおよびＡＯＳのそれぞれは、初期化時またはコンポーネントマネージャを介して新しい構成要素を追加する際に同期サーバがロードするスタンドアローンＤＬＬとして実行される。

【００５６】各同期サーバは、単一アプリケーション専用なものとなっている様子が示されている。しかしながら、同期サーバは、多数の装置タイプを扱うことができる。

【００５７】図９Ｂに示す実施形態では、装置タイプに依存して、ＡＯＳおよびＡＯについて様々な構成がとられる。例えば、Palm（登録商標）ＡＯデータ格納部１０５０は、Palm（登録商標）装置８０４ａ自体に存在し、個別ＡＯＳデータ格納部１０５２は、この構成（Oracleデータベース）のために存在する。ポータル１の場合には、ＡＯＳおよびＡＯはデータ格納部１０５４を用いる。

【００５８】装置エンジンは、他のシステムにおける同期問題を解決することを意図したデータパッケージを追加生成することができる。例えば、競合解決モジュール９４０との間でインターフェイスをとる際に、ユーザは、自分のPalm（登録商標）pilot上のアプリケーションオブジェクトにおける特定データ格納部に変更を施した後、自分のパーソナルコンピュータにおける個人情報マネージャ（ＰＩＭ）アプリケーションにさらに変更を施す場合には、このユーザは、Δエンジンおよびバージョン情報により２つの装置の間における競合が検出された際には、このパーソナルコンピュータになされる変更が「勝つ（win）」ということを指定することができる。これは、本質的には、ある特定セットのデータが正しいものであり、この特定セットのデータが第２セットのデータに置き換わるということを定義している。

【００５９】図１０は、例えばMicrosoft社のWindows（登録商標）をベースとしたオペレーティングシステム環境で用いられるデスクトップ装置エンジンの一実施形態を示す。Windows（登録商標）オペレーティングシステムは、同期を必要とする少なくとも３つの特定アプリケーションを有する。図１０では、システムは、ブックマーク１０２１、連絡情報１０２２および電子メール１０２３のようなデータを有するNetscape Communicatorアプリケーションと、連絡情報１０２４、日程情報１０２５、電子メール情報１０２６、メモ情報１０２７およびタスク情報１０２８を含むMicrosoft社のOutlookアプリケーション１０４２と、お気に入りデータ１０２９、ファイルシステム情報１０３０および個人情報１０３１を含むWindows（登録商標）オペレーティングシステムと、を備える。

22

【００６０】各特定アプリケーション１０４０、１０４２、１０４４は、関連したアプリケーションオブジェクト１０１０、１０１２、１０１４を有する。各アプリケーションオブジェクトのそれぞれは、図９Ａに示した装置について上記のように説明した内容にかかるデルタモジュールにより使用可能な包括的フォーマットとなっているデルタモジュール９５０に対して、データを戻す。さらに、図１０より、同一の特定サーバ上を起動するアプリケーション間でデータを同期させるために、デルタモジュールをどのように利用できるかを理解できる。よって、装置エンジンは、例えばNetscapeからの連絡情報１０２２とOutlookからの連絡情報１０２４との間でイントラシステムの同期を行う。

【００６１】図１０は、さらに、１つの装置上で起動する全アプリケーションを一体化するべくこの１つの装置に提供すべき任意数の様々なアプリケーションオブジェクトを、装置エンジンに含ませることを可能とする、本発明のシステムのモジュラリティを示している。動作時には、特定システムに装置エンジンをインストールする間において、インストールプログラムは、所定システムに存在しうるアプリケーションオブジェクトを提供するように調整される。例えば、Windows（登録商標）マシーン用のインストールプログラムは、Windows（登録商標）マシーンに存在しうるシステムおよびアプリケーション用の任意数のアプリケーションオブジェクトを持っている。このインストーラは、所定アプリケーションが存在しているかをチェックする。また、このインストーラによって、ユーザは、このインストーラが有しているアプリケーションオブジェクトによるアプリケーションサポート用の通常のデフォルトインストール領域ではない場所にインストールされうる付加的なアプリケーション、または、理由はどうであれ、ユーザがアプリケーションオブジェクトをインストールしたくないものであって、かつ、本発明のシステムの一部を明け渡したくないような非選択の所定アプリケーションを追加することができる。

【００６２】インターネットで実行される同期システムにおいて安全性を確保し特定ユーザの識別を行うために、マネジメントサーバが本発明のシステムに設けられる。このマネジメントサーバは、全ユーザにわたって、装置エンジンのネットワーク全体の作用および特性を制御する集中型サーバである。

【００６３】図１４は、本発明の一システムに統合されるマネジメントサーバ１４１０の概略図を示す。また、図１４には、マネジメントサーバ１４１０、格納サーバ１４１５および包括的ＦＴＰサーバ１４２０のすべてに

対するＨＴＴＰリンクを有している装置エンジンの一例が示されている。本発明の処理および図１５から図１７に示すデータの特定インプリメンテーションを参照して後に説明するように、マネジメントサーバは、装置エンジンとの間で相互に作用して、格納サーバ、すなわち、本発明のシステムにかかる装置に特化した情報格納部１４３０にアクセスするための包括的ＦＴＰサーバ１４２０、１４２５における情報に対する許可されたアクセスを制御する。これにより、インターネットに接続されている任意の装置は、マネジメントプロトコルに対してアクセスすることができ、かつ、本発明のシステムにより同期されているデータがアクセスする必要のある、全プラットフォームにわたるユーザ情報を保持することができる。マネジメントサーバは、好ましくは、安全性を確保するために、ＳＳＬ（secure socket layer）を用いて実行されうるハイパーテキスト転送プロトコル（ＨＴＴＰ）を用いて、通信を行う。マネジメントサーバは、同期の実行前に、各装置エンジンにこのマネジメントサーバを用いて認証させる必要のある、認証インターフェイスをサポートする。所定の格納サーバを使用する際には、多数の装置エンジン用の格納部に対する読み込みおよび書き込みアクセスを制御するために、ロッキング意味論（Locking semantics）を利用することができる。例えば、包括的なＦＴＰ要求では、２つの装置エンジンが同一データに対して同時に接続しようとした場合、同一データに対して同時にアクセスする装置エンジンを妨げるためには、ある形式のロッキング制御がなければならない。この場合には、マネジメントサーバは、装置エンジンの取得および更新、ならびに、ネットワークに格納されたデータに対するロックの解除を制御する。

【００６４】各装置エンジンは、マネジメントサーバにより、個別に識別され調査される。これにより、マネジメントサーバと、特定タイプの格納システムおよび装置エンジン構成要素との間における調整作用が考慮される。装置エンジン構成要素のすべては、タグが付けられ、かつ、マネジメントサーバを介してマネジメントに対するバージョンが付与される。

【００６５】装置を実行させることにより、各装置エンジン構成要素の更新済コピーを要求して、装置エンジンシステムの自己更新および再構成を行うことができる。これにより、後に付加的に要求された構成要素を装置エンジンがダウンロードすることを可能とする低バンド幅接続がなされた装置エンジンに対して、最小限のダウンロードをデザインすることができる。

【００６６】システムのさらに別の態様では、マネジメントサーバがクライアントの広告メカニズムをサポートできるような場所に、付加価値構成要素が設けられているので、ウェブブラウザを必要とすることなく、装置エンジンシステム上にバナーまたは同様の広告を表示することができる。広告、統計収集およびこれらと同等なものを循環させることおは、マネジメントサーバのプロトコルを介して処理される。オンライン購入および購読メカニズムもまた、マネジメントサーバのプロトコルを用いてサポートされる。

【００６７】マネジメントサーバは、さらに、システムにおける各ユーザに対して、アカウンティング、サインアップ登録、装置の編集（device selection）、格納サーバの選択（storage server selection）および同様な機能をサポートしている。一実施形態では、マネジメントサーバは、所定ユーザのアカウントについて、パスワード情報および暗号情報を保持することができる。第２実施形態では、このような情報は保持されない。この第２実施形態によれば、マネジメントサーバを維持装置がユーザのアカウントにおけるデータにアクセスするためのパスワードを保持していなければ、ユーザがさらに安心感を持つことができる、という有利な点がもたらされる。

【００６８】さらに、マネジメントサーバに関連する情報、および、マネジメントサーバから本発明のシステムにおける他の構成要素に対するデータのフローについては、図１５から図１７に示す処理フロー図およびデータフロー図を説明することにより明らかとなる。

【００６９】図１７は、データフローの概略図および本発明に基づいて用いられるマネジメントサーバの機能的仕様を示す。

【００７０】図１７に示すように、ウェルカム要求１７１０に引き続いて、ユーザは、追加ユーザモジュール１７１２を使用可能（イネーブル）にし、この後、追加装置モジュール１７１４を使用可能（イネーブル）にするサインアウトを行うことが許される。サインアップが要求されなければ、情報は、モジュール１７１８を介して供給される。

【００７１】図１７に示すように、追加ユーザモジュール１７１２は、装置データベース１７５０におけるユーザに対してユーザレコードを追加する。そのうえ、追加装置モジュール１７１４は、ユーザ装置データベース１７５０に対してユーザおよび装置を追加する。装置リスト１７２０、ならびに、装置エンジンダウンロードおよび更新データベース１７２２は、追加装置モジュール１７１４に対して選択データを供給する。アカウント認証モジュール１７２４は、１７１０におけるウェルカムスクリーンでのユーザログイン、および、追加装置モジュール１７１４の両方からの直接入力を受信する。

【００７２】一旦アカウントが認証および確認されると、１７７０における私用データ格納部を有する本発明のシステムの管理者は、ファイル１７５６、電子メール１７５８、カレンダー１７６０、連絡情報１７６２、メモ１７６４およびタスク１７６６のようなユーザのレコードに対してアクセスすることができるウェブデスクトップ１７５４の提供を選択することができる。情報は、

上述したような本発明のシステムに基づいて同期するプロバイダーデータベース１７５２から選別される。本質的には、プロバイダデータベース１７５２は、上述したような、格納サーバと、各装置エンジン１７８５と、設定データベース１７８７とを含む装置エンジン１７８０からのデータに対してアクセスする。

【００７３】マネジメントサーバの別の部分には、同期を開始し（１７３２）、同期をカウントし（１７３４）、同期を終了させ（１７３６）るためのロッキングモジュールと、ユーザを変更し（１７４２）、装置を追加し（１７４４）、装置を除去し（１７４６）および装置を変更（１７４８）することを含んだ、ユーザ情報を更新するためのロッキングモジュールと、が含まれる。

【００７４】図１４には、格納サーバ１４１５の一例が示されている。格納サーバは、本発明に基づいて、任意数の標準インターネットプロトコルによってアクセス可能な包括的格納モデルを含む一方、本発明のシステムを様々な方法で標準的に実行させることが可能なフレキシブル格納構造が設けられている。これにより、新しいサーバアプリケーションをインストールすることなく、ネットワークサービスを実現することができるとともに、多数の装置エンジンの間における変更情報を矛盾のない方法で通信することが可能となる。

【００７５】１つ以上の格納サーバ１４１５は、装置の収集の間でトランザクションを通信するために用いられうる。ユーザの個人情報ネットワークのそれぞれは、そのネットワーク各自のデータパッケージ格納部内の固有アカウントにより表される。格納サーバ１４１５は、データパッケージの継続性（persistent）格納収集を保持する。この格納収集は、最低でも、ユーザの所定情報ネットワークにおける最も旧式なシステムを同期させることができる程度に十分なデータパッケージであるか、または、ネットワークに供給される新しい装置に情報を追加することができる程度に十分なデータパッケージである。付加的なデータパッケージは、前バージョンの情報をロールバックすることができるように維持されうる。格納サーバは、自動的に、古い方のデータパッケージの記憶を処分し、使われていないアカウントの経年変化をサポートする。

【００７６】各格納サーバ１４１５は、任意のオペレーティングシステムプラットフォーム用の標準ＦＴＰサーバを含む様々なインプリメンテーションを用いて実現されうる。格納サーバは、効率を向上させかつファイアウォールを避けるためのＨＴＴＰプロトコルを用いて実現されうる。格納サーバは、データベースアクセスまたはユーザの全ファイルシステムツリーの単一ファイル記憶のようなローカル記憶用の技術を用いて実現されうる。格納サーバ１４１５は、データパッケージのコピーをシステムにおける別の格納サーバに移動させるための、格納された外部プロトコルモデルを用いることができる。

一実施形態では、格納サーバは、ファイアウォールが親プロトコル（originating protocol）を妨げるような場合には、他の格納サーバに対して別のプロトコルを用いることにより、情報のトンネリング（tunneling）を可能とする。例えば、格納サーバは、ＨＴＴＰプロトコル内部のＦＴＰトラフィックを中継することができる。格納サーバは、別々のマネジメントサーバを必要とせずに、同一のサーバに対する多数の装置エンジンによるアクセスを仲裁するために、各自のロッキング意味論を含むことができる。たとえ格納サーバ１４１５が非常に多くのユーザにわたる大量のデータパッケージを保持することができても、各装置エンジンは、特定ユーザのデータパッケージ記憶領域のみに対してアクセスすることが可能である。これにより、ファイルシステム技術を用いて格納サーバが実現される際には、増加した位取り（scaling）を考慮することができる。

【００７７】一態様として、格納サーバは、各動作について標準ＦＴＰまたはＨＴＴＰ接続を用いて実現される。ＨＴＴＰは、要求と応答とをペアで備えている。すべての要求はコマンドを通知することになっている。パラメータは、"application/X-www-form-URLENCODED"のような一般的な形式で設定されうる。符号化はＲＦＣ１８６６に明記されている。格納サーバ用の機能は、格納サーバがシンプルテキスト列を検索する他のユーザにたどり着くことができるかどうかをテストすることと、バイナリストリームの副次物に存在するようなファイルの内容を転送する"get"コマンドと、格納サーバに対するバイナリストリームのデータのようなputコマンドと、ディレクトリリスティングコマンドと、削除コマンドと、改名コマンドと、存在するコマンドと、これらと同等のコマンドと、を含む。

【００７８】図１５は、本発明にかかる「プル（pull）」同期処理を表現している。図１５に示すプル同期および図１６に示すプッシュ同期の両方は、装置エンジンの観点から実行される。図１５に示したようなプル同期は、好ましくは、プッシュ同期の前に実行される。これにより、装置エンジンは、独自のデータを同期させる必要があるか否かを認識することができる。

【００７９】各装置は、同期を開始するための独自のトリガメカニズムを有する。Windows（登録商標）のクライアントおよびPalm（登録商標）pilotのようないくつかの装置は、ユーザが「同期」ボタンを押圧した際に手動によりトリガされる。携帯電話のような他の装置は、他の装置が同期を完了させた後に、自動的にトリガされる。規則的なタイムベーストリガも同様にサポートされる。ウェブをベースにしたアプリケーションポータルは、ユーザがウェブサイト安全性認証メカニズムにログインする際に同期し、選択的には、ユーザのログアウトまたはセッションタイムアウトにより同期を行うこともできるが、これは、そのユーザがセッションの間におい

てデータを変更した場合に限られる。各同期について、トリガが発生した場合には、どのアプリケーションタイプが装置のために同期を行うべきかが指定される。これにより、トリガの発生により、特定アプリケーションタイプのための同期のみがトリガされうる。マネジメントサーバは、格納サーバに対するトラフィックを最小化するために、特定タイプのアプリケーションに対する同期が必要でないことを指定することができる。同期はサーバに対するＨＴＴＰ要求によりトリガされうる。この要求は、どの装置が同期の対象となるのかについての情報、ならびに、認証および確認（validation）用のマネジメントサーバに送られるユーザログイン情報を保持している。ＨＴＴＰ要求をサーバに対して送信し、この要求のデータ部分における認証情報をマネジメントサーバに対して送信することにより、同期がトリガされる。各装置は、サーバに対して同期要求を出す前に、要求を検索しこの要求の適切なフォーマットを確保することを担うservletを含む。

【００８０】装置の名前および装置のクラスは、同期されている特定の装置タイプを独自に識別し、マネジメントサーバに含まれている。各ユーザは、マネジメントサーバ認証レコードにおいて１つ以上の装置エンティティを有しており、各装置の名前は、このユーザの空間において固有なものとなっている。例えば、ユーザが自分の個人識別番号を持った５つの装置を持っている場合、５つの認証レコードが存在する。これらは、２つのWindows（登録商標）装置、２つの異なるPalm（登録商標）装置、１つのウェブサービスポータルであるかもしれないが、これらはそれぞれ、独自の個人識別番号を有している。

【００８１】図１５に示すように、プル同期処理は、上述したトリガの発生が同期要求をトリガする際に、アイドル状態１４０５で開始する。同期要求は１４１０で確認され、この要求が照合されれば、工程１４１５において格納サーバに対する接続がなされる。一旦接続が確立されると、マネジメントサーバを介してユーザ識別を認証するために、工程１４２０においてマネジメントサーバに対する接続がなされる。認証が成功すると、マネジメントサーバは、競合する装置エンジンが同一データに対して同時に接続することができないように、格納サーバにおいてマネジメントサーバロックを開始する。工程１４１０－１４２５のどこかで失敗が生ずると、システムはアイドル状態１４０５に戻る。一旦エンジンサーバロックが必要になると、格納サーバ上に新バージョンのデータが存在するか否かを決定するために、格納サーバは工程１４３０においてチェックされる。新バージョンが存在しなければ、同期処理は終了する。

【００８２】新バージョンのデータが存在すれば、装置エンジンは、工程１４３５において、相違情報"o get Δ"を検索する。

【００８３】一旦Δが検索されると、競合は工程１４５０において解決される。競合解決工程によって、ユーザは、装置のサーバ部分およびローカルデータの両方に対して変更がなされている多数のタイプのデータに対する競合を解決することができる。

【００８４】一旦競合が工程１４５０において解決されると、Δ′sが工程１４５５で付与される。Δ付与工程１４５５は、フィルタおよびマッピングがシステムのローカル装置エンジン側に報告されるようにする。工程１４６０、１４６５、１４７０および１４７５に示すように、Δは、アイテムレベル１４６０、アプリケーションレベル１４６５、装置レベル１４７０、または、ネットワークレベル１４７５における更新を含む。上述した工程のそれぞれにおいて、Δ検索工程１４３５へ戻るループが設けられている。もはやΔ′sが利用可能でなければ、マネジメントサーバロックは、工程１４４０において解除される。

【００８５】図１６は、本発明のシステムおよび方法にかかるプッシュ同期の一例を示す。アイドル状態１５０５から始まって、工程１５１０で確認されれば同期が発生し、工程１５１５においてΔ′sがチェックされる。どのタイプの変更事項が発生したかによって、ネットワークΔ１５２０、装置Δ１５２５、場所Δ１５３０またはアイテムΔ１５３５が生成される。

【００８６】一旦所定アプリケーションに対してΔ′sが生成されると、本発明にかかる方法は、格納サーバに対する接続を可能にする工程１５４０にとどまる。格納サーバに対する接続がなされている際には、システムにおけるユーザを認証するために、マネジメントサーバ１５４５に対してさらなる接続が発生しうる。上述した時点のどこかで失敗が生ずると、処理はアイドル状態１５０５に戻ることになる。認証時には、マネジメントサーバロックは、多数の装置エンジンが同一データに対して同時に接続することのないように、イネーブルとされる。

【００８７】一旦工程１５５５でロックがかけられると、Δ′sがシステムにアップロードされる。図示しているように、この工程１５５５は、アイテムΔ１５７５およびアプリケーションΔ１５７０をアップロードすること、装置Δ１５６５またはネットワークΔ１５６０をアップロードすることを含む。一旦Δ′sがサーバにアップロードされてしまうと、マネジメントロックサーバ１５８０が解放され、格納サーバに対する接続が工程１５８５において終了する。

【００８８】このようなプッシュ同期は、サーバに直接発生する必要はないが、図１から図７に示した本発明の多数の実施形態で説明した内容にかかる第２の装置エンジンに直接生じうる。

【００８９】一旦汎用データフォーマットに情報が供給されると、装置エンジンはこのフォーマットをデータパッケージ内に編成する。これにより、各データパッケー

29

ジは、特定アプリケーションについての任意かつすべての情報に対する変更事項の記述を含み、データパッケージの収集によって、あらゆる異なるタイプのデータを含む全装置エンジンにわたる変更事項を記述される。暗号化および圧縮化に伴って、データパッケージは非常にコンパクトなものとなるので、本発明のシステムにわたってバンド幅および記憶必要条件を最小にすることができる。

【００９０】本発明の一特定態様として、データパッケージを暗号化することは、ストリーミングフォーマットに設けられうるので、装置エンジンレベルで、最小限の記憶容量およびメモリ構成を有する装置エンジンによる処理を可能とする。

【００９１】装置エンジンは、ストリームを読み込み、この装置エンジンが設けられているシステムに存在する特定情報を更新するために、この装置エンジンがどのアプリケーションのどのレコードが必要であるのかを、決定する。

【００９２】データパッケージは、バイナリデータフォーマットで供給されうる。これにより、データパッケージは、バイトレベルで非アプリケーションデータに対する変更事項を暗号化することができる。よって、システムにおける単一ビットが変更した場合、本発明のシステムは、別のシステムにおけるそのビットを同期させることができる。変更事項は、一連のバイトレベル変更動作として、記述される。このような暗号化の１つは、一連の挿入およびコピー動作を用いている。挿入およびコピーの動作は、通常、ソースファイルから多くのバイトを特定の方法により「挿入する」こと、次いで、変更されたソースをどれくらいのバイトだけ特定ファイルに挿入しなくてはならないのか、次いで、特定の新ファイルからどれくらいのバイトだけ挿入するのか、を定めているので、相違エンジンは、ストリームの中からバイトを取りだし、この取り出したバイトを新ファイルに挿入して、このファイルの新バージョンを生成する。

【００９３】当業者であれば容易に理解できることであるが、これにより、ユーザは、例えば、ワードプロセッサ文書またはその他のタイプの添付物のようなバイナリファイルを変更し、このような添付物をバイナリレベルで同期させることができる。具体的には、ある第１の人物が、ワード文書の電子メールを第２の人物に転送したならば、この第２の人物は、この文書を修正し、修正したこの文書を第１の人物に送り返したいと思う。なぜならば、第１の人物は、自分のシステム上にオリジナルファイルを持っており、両方のシステムが本発明のシステムにおいてイネーブル（使用可能）とされれば、第２のシステムは、第２のユーザにより意図されたような文書を作成するべく変更データを用いて第１のシステムが第２のシステム上で文書を再構築することができるようにするために、第１のシステムに対して変更事項または相違情報を送信するだけでよいからである。

30

【００９４】本発明のシステムに基づいて通信イシュー（Communication issue）を取り扱うために、データパッケージの世代およびアプリケーションをマルチキャッシングすることが利用される。さらに、データパッケージがこれより大きなメタデータパッケージに合併されうるということを理解されたい。多数の装置パッケージの編成のような、このようなメタデータ情報は、より大きなシステムパッケージに暗号化される。各システムパッケージは、本質的には、データパッケージの暗号化されたシーケンスである。

【００９５】図１２は、本発明に基づいて用いられる、データパッケージの一般的フォーマット、および、オブジェクトストリーム階層構造の汎用データフォーマットを示す。図１１および図１２を参照するに、特定アプリケーションデータ構造における各アイテムが、図１３に示すような、ファイル、フォルダ、連絡情報、電子メール、カレンダー等のような特定の分類を有していることは明らかである。汎用データ構造は、システムによりサポートされた各アプリケーションから、各タイプのデータについてマッピングされたアイテムフィールドを含んでいる。よって、各データフィールドマッピングの「マスター」リストは、多数のアイテムを含む。各アプリケーションオブジェクトは、このようなフィールドのサブセットを必要とする。他のウェブポータルを含む、あらゆる装置上で利用可能なあらゆる情報に対してアクセスできるようにするウェブポータルアプリケーションのために用いられるアプリケーションオブジェクトが１つの例外となっている。

【００９６】所定アイテム１２５０用に含まれるアイテムフィールド１２６０の特定例が図１３に示されている。これらの例示アイテムオブジェクトは、例えばMicrosoft社のOutlookのような割り当てからものである。Outlookは、メモアイテム１３１０、電子メールアイテム１３２０、タスクアイテム１３３０、カレンダーアイテム１３４０、ブックマークアイテム１３５０、ファイルアイテム１３６０、チャネルアイテム１３７０、フォルダアイテム１３８０および連絡情報アイテム１３９０を準備しており、また、これらのすべては、図１３に示したようなフィールドを有する。

【００９７】データフォーマットは、また、アイテムの分類と、この後特定カテゴリに続くこれらのアイテムに関連したアイテムフィールドとを与える、フォルダ情報１２４０を含む。

【００９８】アプリケーションオブジェクト１２３０は、ストリームにおける情報が含まれたものからの、様々なタイプのアプリケーション上の情報を含む。装置オブジェクト１２２０は、この情報の発信源である、元のタイプの装置上の情報を含む。ネットワークオブジェクト１２１０は、データストリームにおける情報が特定ユ

31

ーザから送られたものであるということを定めるための、ユーザレベル上の情報を含む。

【００９９】上述したように、各アプリケーションオブジェクトは、フォルダレベル上の情報の収集を管理することを可能とし、かつ、情報のフォルダ階層を管理することを可能とする、フォルダ記憶インターフェイスをサポートする。このアプリケーションオブジェクトは、また、レコードもしくはファイルのような各情報エンティティ、または、レコード内のフィールドのような情報エンティティの構成要素を許容するアイテムインターフェイスを含む。さらに、各アプリケーションオブジェクトは、コンピュータメーカのアプリケーションを検出するためのインターフェイスをサポートしている。

【０１００】データパックは、本質的には、情報に対する変更事項を記述した一連のトランザクションを含んでいる。この情報は、２つの基本的なタイプ、すなわち、構造的データすなわちアプリケーションデータと、非構造的データすなわちバイナリファイルデータと、に及ぶ。トランザクションは、実際の内容オブジェクトを表現するために、タグを有した効果的なストリーミングフォーマットを用いて暗号化される。この技術によれば、新しい内容がサポートされた際に、データパックフォーマットを連続的に拡張させることができる。

【０１０１】パッケージの一般的構造は、データパッケージに含まれることになる、アプリケーションデータ、ファイルデータ、ファイル、オブジェクトおよび識別子を規定する。通常、トランザクション、アプリケーションデータ、ファイルデータおよびファイルは、事前に記述されている。

【０１０２】データパッケージの第１部分は、このデータパッケージの識別子である。各トランザクションは、オブジェクトおよび動作の基本的構造を有する。各内容は、オブジェクトと称され、ＵＵＩＤ（Universally Unique Identifier）を用いて個別に表現される。オブジェクトは、一般的には、動的に生成されたＵＵＩＤにより表現されるが、これよりも共通のオブジェクトは、静的ＵＵＩＤにより表現される。各ＵＵＩＤは、好ましくは、システムプロバイダにより割り当てられた固有の１２８ビット値を有する。

【０１０３】トランザクションは、個別ファイルの形をとる処理しやすいブロックに解体される。この後、これらのファイルは、選択的に、圧縮され、暗号化され、および、適当なヘッダが先頭に付与される。トランザクションは、以下に示す規則に基づいて、特定ファイルに分類される。

【０１０４】・アカウント情報に関連したトランザクションは、データパックファイルに分類される。

【０１０５】・特定データクラスに関連するトランザクションは、データパックファイルに分類される。

【０１０６】・バイナリデータに関連するトランザクション

32

は、各ファイルオブジェクト毎に別々のデータパックファイルに分類される。

【０１０７】データパックファイルは、ファイルネームに基づく特定規則を用いて識別される。形式"UUID.VER"というファイルネームについては、ＵＵＩＤが特定オブジェクト用の識別子であり、ＶＥＲはトランザクションバージョン番号である。このバージョン番号は、大きなバージョン番号用に用いられる付加ディジットを用いた"D0001"の形式をとる。この"D000"値は、好ましくは、オブジェクト用ベースバージョンのために確保されている。

【０１０８】ユーザアカウント用のＵＵＩＤは、マネジメントサーバ（ＭＳ）により生成される。ＭＳは、また、ユーザアカウント内のデータパックファイルを理解するためのルート構造を与える、ＵＵＩＤ値およびバージョン番号の現在のテーブルを保持する。ＭＳは、また、多数の装置エンジンが同期をしようとする際に整合性を維持するのに必要なロッキング意味論を供給する。

【０１０９】すべてのデータパックは、データパックに関する基本的な内容の情報を与える標準ヘッダが先頭に付与される。圧縮化および暗号化ヘッダは、必要とあれば、データパックヘッダの後に付与される。

【０１１０】データパッケージヘッダ情報は、バージョンシグネチャ、付与されたバージョン情報、コンテンツタイプ、Δエンジンタイプ、圧縮タイプ、暗号タイプ、付与されたサイズ、暗号化されたサイズ、圧縮化されたサイズ、ローデータのサイズ、および、アプリケーション用の使用可能なフォーマットにデータを変換するべくこのデータを解読化する際に、装置エンジンに用いられるその他のデータを、含んでいる。

【０１１１】ヘッダのフォーマットは好ましくは次のようになる。

【０１１２】

【表１】

タイプ	バイト数
バージョン	4
シグネチャ	4
適用バージョン	8
内容タイプ	4
デルタタイプ	4
圧縮化タイプ	4
暗号化タイプ	4
適用サイズ	4
暗号化サイズ	4
圧縮化サイズ	4
ローサイズ	4
予約	TBD

以下のようコンテンツタイプ（ContentType）値が利用可能である。

【０１１３】

【表２】

33

フィールド	コメント
DP_CONTENT_RAW	ロー
DP_CONTENT_COMPRESSED	圧縮化
DP_CONTENT_ENCRYPT	暗号化
ED	

デルタタイプ（DeltaType）は、用いられるバイナリフ ＊

フィールド	コメント
PackageDeltaTypeUninitialized	未初期化
PakageDeltaTypeRawData	ローバイナリデータ
PackageDeltaTypeDeltaXDelta	Xデルタバイナリ相違
PackageDeltaTypeDeltaBDiff	Bdiff バイナリ相違

圧縮タイプは、データパックが圧縮化されているかどうかを明記する。圧縮タイプが明記されていれば、データパック圧縮ヘッダがデータパックヘッダの後に続く。以下に示すような圧縮タイプ（CompressionType）値は、 ※

フィールド	コメント
PackageCompressionTypeUninitialized	未初期化
PackageCompressionTypeNone	なし
PackageCompressionTypePK	PKZipフォーマット
PackageCompressionTypeLZS	LZSフォーマット

暗号タイプは、データパックが暗号化されているかどうかを明記する。暗号タイプが明記されていれば、データパック暗号化ヘッダがデータパックヘッダの後に続く。以下に示す暗号タイプ値はデータパック暗号タイプ（Da★

フィールド	コメント
PackageEncryptionTypeUninitialized	未初期化
PackageEncryptionTypeNone	なし
PackageEncryptionTypeXORTest	XORマスクデータ
PackageEncryptionTypeBlowFish	Blowfish
PackageEncryptionTypeTwoFish	Twofish

すべてのデータパック圧縮ヘッダは、以下に示すフォーマットを用いて暗号化される。 ☆

フィールド	サイズ(バイト数)	コメント
Size	4	このヘッダを含むデータのサイズ
Version	4	バージョン(1)
Signature	4	シグネチャ(4271)
HeaderType	4	ヘッダタイプ(HeaderTypeCompression)
Reserved	12	予約
DecompressedSize	4	伸張化サイズ
Reserved	50	予約
Reserved	12	予約

以下に示すヘッダタイプ（HeaderType）値は、データパッケージヘッダタイプ（DataPackageHeaderType）を用いることにより利用可能である。

【０１１８】

【表７】

フィールド	コメント
HeaderTypeUninitialized	未初期化
HeaderTypeEncryption	暗号化ヘッダ
HeadeTypeCompression	圧縮化ヘッダ
HeaderTypeRaw	ローヘッダ

34

＊ァイル相違のタイプを暗号化する。以下に示すデルタタイプ値は、デルタパッケージデルタタイプを用いて利用可能である。

【０１１４】

【表３】

※データパッケージ圧縮タイプ（DataPackageCompressionType）を用いて、利用可能である。

【０１１５】

【表４】

★taPackageEncryptionType）を用いることにより、利用可能である。

【０１１６】

【表５】

☆【０１１７】

【表６】

すべてのデータパック暗号ヘッダは、以下に示すフォーマットを用いて暗号化される。

【０１１９】

【表８】

50

フィールド	サイズ(バイト数)	コメント
Size	4	このヘッダを含むデータのサイズ
Version	4	バージョン(6)
Signature	4	シグネチャ(4270)
HeaderType	4	ヘッダタイプ(HeaderTypeEncryption)
Reserved	12	予約
DecryptedSize	4	解読化サイズ
InitValue	16	TBD
KeyLength	4	TBD
ClearTextKeyBits	4	TBD
Salt	4	TBD
PadBytes	4	TBD
HMAC	20	TBD
Reserved	12	予約

以下に示す動作値は動作クラスを用いることにより利用可能である。

【０１２０】

【表９】

フィールド	コメント
clNop	なし
clAdd	追加
clDelete	消去
clChange	変更
clMove	移動
clRename	リネイム
clForceChange	競合を除く強制変更

＊以下に示すフィールドデータタイプ（FieldDataType）値がデータタイプ（clDataType）を用いることにより利用可能である。

【０１２１】

【表１０】

＊

フィールド	コメント
clInvalidType	TBD
clString	32ビット長のプリフィックスを有するユニコードストリングバイト
clString8	8ビット長のプリフィックスを有するユニコードスプリングバイト
clString16	16ビット長のプリフィックスを有するユニコードスプリングバイト
clEmptyString	TBD
clBlob	バイトストリームの前にある32ビット長
clBlob8	バイトストリームの前にある8ビット長
clBlob16	バイトストリームの前にある16ビット長
clEmptyBlob	TBD
clByte	8ビット値
clShort	16ビット値
clDword	32ビット値
clQword	64ビット値
clDate	日付タイプ(ダブル)
clDouble	8バイトリアル
clFloat	4バイトリアル
clUuid	16バイトuuid
clZero	ゼロ値
clOne	1値
clUnspecified	未指定値
clDefault	デフォルト値
clCollection	32ビット値を有する収集
clCollection8	8ビット値を有する収集
clCollection16	16ビット値を有する収集
clEmptyCollection	長さを有しない収集

データパッケージオブジェクトは、以下に示すような階層に編成される。

Account::= DeviceList + DataClassList

DeviceList::= {Device}

DataClassList::= {DataClass} + ProviderList

ProviderList:: = {Provider} + DataStoreList

DataStoreList:: = {Folder} + ItemList

ItemList::= {Item} + FieldList

FieldList::= {Field}

アカウントは、ユーザアカウントについての情報を認識するルート構造である。このアカウントは、ネーム、パスワード、ユーザネームおよびバージョンのような、具体的なフィールドタグ（eFieldTag_[NAME]）を有することができる。フィールドタグアイテムタイプ値は、加算アイテムタイプを用いて、ItemType_PINのように明記される。

【０１２２】装置は、アカウントの一部分として認識されたシステムである。これの例には、パーソナルコンピュータ、ハンドヘルド、ウェブサイトなどが含まれる。装置は、"name"、"type"、ならびに、ポータル、Palm、Windows（登録商標）、携帯電話といったアイテムタイプ値（eDevice_[Name]）のようなタグ（eField_[Nam

e]）を有することができる。

【０１２３】データクラスは、類似する情報タイプの集合である。多くのデータクラスは、特定アカウントのために表現される。データクラスは、ネーム、アイテムタイプ、サブタイプ、保持中、プロバイダ、フィルタおよ＊

(eDataClass_[Name])：

Tag	記述
UNKNOWN	未知
CONTACT	連絡／アドレスブック
EMAIL	電子メール
CALENDAR	カレンダー
TASK	実行すべきタスク
NOTE	ノート／メモ
JOURNAL	ジャーナル
BROWSER	ウェブブラウザのお気に入り、クッキー等
FILESET	ファイルの収集
PIN	アカウント情報
DEVICE	装置情報
FILEBODY	ファイルの内容

プロバイダは、データクラス内に特定情報を保持するアプリケーションである。特定データクラスには１つ以上のプロバイダが存在しうる。フィールドタグは、ネーム、適用オブジェクトＩＤ、パスワード、ユーザネームおよびバージョンを含む。このプロバイダ（eProvide[Name]）に利用可能なプロバイダタグの例では、ポータル、Palm（登録商標）、MicrosoftOutlook（登録商標）、LotusOrganizer、MicrosoftInternetExplorer、MicrosoftWindows（登録商標）等を含む。

【０１２５】データ記憶部は、プロバイダ内に情報を格納するためのコンテナである。特定プロバイダには１つ以上のデータ格納部が存在しうる。フォルダは、データ格納部内の情報についての構造化組織である。データ格納部は、フォルダをサポートするためには必要とならない。各データ格納部用にサポートされるタグ（eFieldTag_[Name]）は、ネーム、アイテムタイプ、保持中、オリジナルパスを含む。データ格納部のために利用可能なアイテムタイプは、アンノウン、MAPI、データベースおよびStore_Fileを含む。

【０１２６】フォルダは、データ格納部内の情報についての構造的組織を表現する。データ格納部は、フォルダをサポートするために必要とはならない。フォルダは、ＵＵＩＤにより表現され、かつ、以下に示すようなフィールドタグ（eFieldTag_[Name]）のいずれかを含む。すなわち、フィールドタグは、ネーム、アイテムタイプ、保持中、ファイル属性、生成日、変更日、アクセス日および特別フォルダタイプである。

【０１２７】eFieldTag_ItemType値は、加算アイテムタイプを用いて、eItemType_FOLDERのように明記される。

【０１２８】アイテムは、実際のユーザデータからなる個別情報構成要素である。これらのアイテムは、ネー

＊びバージョンのようなタグ（eFieldTag_[Name]）を含むことができる。

【０１２４】以下に示すアイテムタイプ値は、加算データクラス（eDataClass_[Name]）を用いることにより利用可能である。

ム、アイテムタイプ、保持中およびバージョンのようなフィールドタグを含む。

【０１２９】ファイルアイテムは、一般的には、以下のような付加フィールドタグ（eFieldTag_[Name]）を含む。

(eFieldTag_[Name])：

FileAttributes
CreationDate
ModificationDate
AccessDate
FileSize
FileBody
DeltaSize
Hash

アイテムタイプは、フォーマット（eItemType_[Name]）を採用し、拡張（extended）、フォルダ、添付物（attachment）、連絡情報、個別リスト（distlist）、電子メール、カレンダー、タスク、コール（call）、メモ、ポスト（post）、ジャーナル、フォーム、スクリプト（script）、規則、お気に入り（favorites）、サブスクリプション（subscription）、common_favorites、デスクトップ、common_desktop、スタートメニュー、チャネル、クッキー（cookies）、プログラム、common_program、スタートアップ、common_startup、送信（sendto）、現在（recent）、internet_cache、ヒストリ、mapped_drives、プリンタ、ドキュメント、ドキュメントのテンプレート（doctemplate）、フォント、window_settings、app_data_folder、app_settings、ファイルセット（fileset）、ピン（pin）、装置、data_store、ファイル、プロバイダ、data_class、インターナル（internal）を含む。

39　　40

【０１３０】フィールドは、ベースタイプ定義のセットのうちの１つに基づいている。すべてのフィールドタグ情報は、以下に示すフォーマットを用いて暗号化され＊る。

【０１３１】

【表１１】

フィールド	サイズ（バイト数）	コメント
FieldTag	16	固有タグ番号
FieldType	6	フィールドベースタイプ
FieldSubType	10	フィールドサブタイプ

アンノウン（unknown）、ロング（long）、dword、日付（date）、ストリング（string）、バイナリ、フロート（float）、ダブル（double）、収集、ユニークＩＤ（uniqueid）、ＵＵＩＤ、ファイル、無効（invalid）を含む多くのフィールドタイプが利用可能である。ＬＯＮＧは、ビッグエンディアン（big-endian）フォーマットで暗号化された４バイト値である。フィールドタイプＤＷＯＲＤは、ビッグエンディアンフォーマットで暗号化された４バイト値である。フィールドタイプＳｔｒｉｎｇは、１ＮＵＬＬバイトが後に続く一連のユニコードキャラクタである。ＭＢＣＳ値にはインターフェイスが設けられる。フィールドタイプＢｉｎａｒｙは一連のバイトである。フィールドタイプＵｎｉｑｕｅＩＤは、汎用固有識別子（ＵＵＩＤ）標準により定義されるような一連のバイトである。ＡＯインターフェイスが、ローカル個別識別子（ＬＵＩＤ）値に設けられる。フィールドタイプＱＷＯＲＤは、ビッグエンディアンフォーマットで暗号化された８バイト値である。フィールドタイプＦｉｌｅは、ファイルボディデータを含む個別データパックを表現するＵＵＩＤである。ファイルに対して全パスを記述する１ＮＵＬＬバイトが後に続く一連のユニコードキャラクタに、ＡＯインターフェイスが設けられる。任意数のファイルサブタイプが利用可能である。各サブタイプは、サポートされた全ユーザアプリケーションから利用可能な全データタイプを含む。十分に理解できるように、サブタイプの数が、非常に大きくなり、かつ、本発明のシステムによりサポートされる各新アプリケーションが追加されるにつれて変化する、という可能性がある。サブタイプには、例えば以下に示すものが含まれる。

サブフィールドの記述	記述
Base	サブタイプが未指定
EmailAddress	電子メールアドレス
EmailAddressList	電子メールアドレスリスト
SearchKey	検索キー
CategoryList	カテゴリリスト
StringList	ストリングリスト
DistributionList	配布リスト
Gender	ジェンダー（enumGender）
TimeZone	タイムゾーン（enumTimeZone）
Boolean	ブール（TBD）
NonZeroBool	ゼロ値を有するブール（enumNonZeroBool）
Priority	優先度
Sensitivity	感度（enumSensitivity）
Importance	重要度（enumImportance）
SelectedMailingAddr	選択された送信アドレス（enumSelectedMailingA
ddr）	
TaskStatus	タスク状態（enumTaskStatus）
FlagStatus	フラグ状態（enumFlagStatus）
RecurrenceType	繰返しタイプ（enumRecurrenceType）
DayOfWeek	週における日（enumDayOfWeek）
DayOfMonth	月における日（１～３１）
InstanceOfMonth	月の例（enumInstanceOfMonth）
MonthOfYear	年における月（enumMonthOfYear）
BusyStatus	ビジー状態（enumBusyStatus）
AttachmentType	添付物タイプ（enumAttachmentType）
MailBodyType	メール本体タイプ（enumMailBodyType）
RGB	ＲＧＢカラー値

41　　42

ManagedState	処理状態（enumManagedState）
FaoId	プロバイダに対するＦＡＯ－ＩＤ
SpecificationFolderType	特別フォルダタイプ（enumSpecialFolderType）
ResponseState	応答状態（TBD）
ResponseStatus	応答ステータス（TBD）
JournalStatus	ジャーナルステータス
PageStyle	ページスタイル
PageNumberMethod	ページ番号方法
DelegationState	削除状態
MeetingStatus	ミーティングステータス
MeetingInvitation	ミーティング招待
CalendarType	カレンダータイプ
DateOnly	日付のみ
TimeOnly	時間のみ
PhoneNumber	電話番号
URL	ＵＲＬ
FilePath	ファイルパス
PopMessageID	ＰＯＰメッセージＩＤ
MIMEType	ＭＩＭＥタイプ
INVALID	すべての値はこれ以下でなくてはならない

図１８は、上述した様々な構成要素を含む、データ転送および同期システムの一例の一般化されたブロック図である。具体的には、このシステムは図７に示したようなネットワーク７００を含む。ネットワーク７００は、図７における格納サーバ３００$_1$、３００$_2$のような１つ以上の格納媒体を含む。図７に示すような複数の装置は、ネットワークに接続することが可能なものであり、かつ、上述した技術を用いて、オフライン方法により同期情報を交換することが可能なものである。これらの装置は、家庭用ＰＣ７１０およびオフィス用ＰＣ７０２を含み、これらは両方とも上述した技術を用いてお互いに同期することが可能である。上述したような装置エンジンは、様々な装置とネットワーク７００との間に接続される。

【０１３２】図１８において、クライアントソフトウェアは、家庭用ＰＣ７１０とオフィス用ＰＣ７０２の両方にインストールされており、Microsoft社のWindows（登録商標）のようなオペレーティングシステムに関連して動作するように構成されている。実行時には、クライアントソフトウェアは、ユーザＰＣ上の様々なアプリケーションと相互に作用を及ぼす。ユーザは、クライアントソフトウェアと対話し、アプリケーションが優先されるようにこのソフトウェアを構成する。この後、データが、様々なアプリケーションから抽出され、このデータが生成された特定アプリケーションおよび装置とは関係のないフォーマットで編成され、データパッケージに組み込まれる。本発明の具体的な実施形態では、連絡情報、ブックマークおよび日程行事を含む様々なクラスのデータが、この方法により処理される。

【０１３３】一実施形態では、Microsoft社のOutlookが図１８における家庭用ＰＣ７１０にインストールされる。この例では、１０個の連絡情報がOutlookでプログラムされる。ユーザは、クライアントソフトウェアに対してOutlookを用いてこれらの連絡情報を同期させるように指示する。クライアントソフトウェアは、Outlookにアクセスし、上記１０個の連絡情報を抽出し、この連絡情報をデータパックCONT.DOOOに統合する。ここで、ＵＵＩＤ“ＣＯＮＴ”は特定オブジェクトのような連絡情報を示し、“Ｄ０００”はこのデータパックがバージョン「０」であることを示している。これらの連絡情報は、それぞれが固有のＩＤ番号１、２、...１０を割り当てられた１０個のトランザクションとして、データパックCONT.DOOOの中に結合される。例えば、ＩＤ２は、John Smith用の連絡情報を示す。この例では、各トランザクション１、２、...１０は関連したアクション（associated action）である「追加」を有する。この後、データパックCONT.DOOOは、ネットワーク７００にアップロードされ、格納サーバ３００$_2$に格納される。

【０１３４】この後、オフィス用ＰＣ７０２は、ネットワークに接続し、データパックCONT.DOOOを識別する。具体的には、このような識別には、オフィス用ＰＣ７０２がマネジメントサーバ１８０２に対して信号を送信すること、この例では、オフィスＰＣ７０２が特定データクラス（この例では連絡情報）用のいかなるデータパックをもダウンロードしていないということをマネジメントサーバ１８０２に通知することが含まれる。マネジメントサーバ１８０２は、オフィス用ＰＣ７０２に対して、オフィス用ＰＣ７０２がネットワーク７００に前回接続して以来、連絡情報用変更情報のデータパッケージがサーバ３００$_2$に格納されていないことを示す信号を

43

送信して応答する。これに応えて、オフィスＰＣ７０２は、マネジメントサーバ１８０２に対して、データパッケージを要求する信号を送信する。サーバ３００$_1$に格納されている最新のデータパッケージは、この例では、バージョン０の連絡情報であるCONT.D000と識別される。このデータパックCONT.D000は、この後、オフィス用ＰＣ７０２にダウンロードされる。このデータパックにおける変更情報、この例では、連絡情報における「追加」が、この後、オフィス用ＰＣ７０２における適切なアプリケーションに適用される。この例では、オフィス用ＰＣ上のクライアントソフトウェアは、Lotus Notesアプリケーションを用いて連絡情報を同期させるように構成されている。この結果、CONT.D000からの１０個の追加は、Lotus Notesの連絡情報に適用されるので、家庭用ＰＣ７１０およびオフィス用ＰＣ７０２における連絡情報は同期する。オフィス用ＰＣ７０２は、この後、オフィス用ＰＣ７０２は、バージョン番号０の連絡情報データパッケージをオフィス用ＰＣ７０２が適用したことを示す信号を、マネジメントサーバ１８０２に返送する。この情報は、好ましくは、ネットワークに接続されている全装置のそれぞれが変更情報をダウンロードおよびアップロードすることができるように、マネジメントサーバ１８０２によりレジストリ１８０４に保持される。

【０１３５】この後、オフィス用ＰＣ７０２のユーザは、Lotus Notesの連絡情報を更新し、１つ以上の連絡情報を追加する。この例では、２０の連絡情報が追加される。この後、Lotus Notesアプリケーションは、ネットワーク７００に対して第２のデータパッケージをアップロードする。この第２のデータパッケージは２０個の連絡情報を含んでおり、それぞれの連絡情報は関連するアクション「追加」を有する。このデータパッケージは、勿論、CONT.D000における情報よりも新しい変更情報を表現する。オフィス用ＰＣ７０２によりアップロードされたデータパッケージは、固有のファイルネーム、この例ではCONT.D001により識別される。加えて、オフィス用ＰＣ７０２は、マネジメントサーバ１８０２に対して、CONT.D001がネットワーク７００にアップロードされたことを確認する信号を送信する。オフィス用ＰＣ７０２が、現行であること、すなわち、CONT.D001における変更情報を既に適用していることを示すために、レジストリ１８０４がアップロードされる。

【０１３６】この後、家庭用ＰＣ７１０はネットワーク７００に接続し、家庭用ＰＣ７１０におけるクライアントソフトウェアは、ネットワーク７００に接続されたマネジメントサーバ１８０２と通信を行う。具体的には、家庭用ＰＣは、家庭用ＰＣ７１９がデータネットワーク７００に前回接続して以来の、最新バージョンの変更情報としてのCONT.D000を識別する信号を、マネジメントサーバ１８０２に対して送信する。マネジメントサーバ

44

１８０２は、格納サーバに対して、もっと新しい、連絡情報に対する変更情報のデータパッケージを問い合わせる。マネジメントサーバ１８０２は、このようなデータパッケージ、この例ではCONT.D001を識別し、家庭用ＰＣ７１０に対してそのようなデータパッケージが存在することを通知する情報を、家庭用ＰＣ７１０に対して送信する。この後、家庭用ＰＣ７１０におけるクライアントソフトウェアは、新しいデータパッケージを要求した後、マネジメントサーバ１８０２は、このデータパッケージを家庭用ＰＣ７１０に対してダウンロードする。この後、このデータパッケージにおける変更情報、この場合には追加すべきCONT.D000からの２０個の連絡情報は、家庭用ＰＣ７１０におけるMicrosoft社のOutlookにより保持されている連絡情報に適用される。Microsoft社のOutlookに対して変更情報を通信すること、および、この後Outlookにおける連絡情報を更新することは、家庭用ＰＣ７１０におけるクライアントソフトウェアにより調整される。この後、家庭用ＰＣ７１０およびオフィス用ＰＣ７０２における連絡情報は、再度同期する。

【０１３７】「追加」だけでなく他のトランザクションが、本発明の一実施形態に提供される。これらのトランザクションのうちの１つは、「変更」トランザクションである。この例を用いると、CONT.D001がネットワーク７００にアップロードされた後、ある人物についての連絡情報が変更されることがある。例えば、John Smithは、電話でユーザに電話し、このユーザに対して自分の電話番号が変わったことを教えることがある。この後、このユーザは、オフィス用ＰＣ７１０にアクセスして、自分の連絡情報におけるJohn Smithの電話番号を変更する。

【０１３８】この後、上記ユーザは、例えば、クライアントソフトウェアによるコンピュータスクリーンに表示された「同期」ボタンを適用するので、新データパッケージすなわち変更ログCONT.D002が生成されネットワーク７００にアップロードされ格納サーバ３００$_1$、３００$_2$のうちの１つに格納される。マネジメントサーバ１８０２に対してデータパッケージCONT.D002がアップロードされたことを通知する、オフィス用ＰＣ７１０によりマネジメントサーバ１８０２に対して送信された信号がアップロードされる。データパッケージCONT.D002は、アクション「変更」が「追加」に代えて用いられている点において、データパッケージCONT.D000およびCONT.D001と異なっている。変更コマンドおよび関連する変更情報は、特定ユーザに関連している。具体的には、変更指示は、適切なＩＤ、この場合にはJohn Smithを表すＩＤ番号２に関連している。加えて、データパッケージCONT.D002は、変更すべきフィールド、この場合には「電話」およびその新しい情報（この場合にはJohn Smithの新しい電話番号）を含む。

45

【０１３９】この結果、家庭用ＰＣ７０２が上述した技術を用いてネットワーク７００に接続する際には、データパッケージCONT.D002が家庭用ＰＣ７０２にダウンロードされ、クライアントソフトウェアは、ＩＤ番号２について、フィールド「電話」内の情報が更新されていることを認識する。家庭用ＰＣがネットワーク７００に次回接続する際には、家庭用ＰＣ７０２は、この家庭用ＰＣ７０２がCONT.D002を受信したことを示す信号をマネジメントサーバに送信する。この後、Microsoft社のOutlookによりこの連絡情報に対して変更がなされる。家庭用ＰＣ７０２は、この後、家庭用ＰＣ７０２がバージョン番号２の連絡情報データパッケージにおける変更情報を受信して適用したことを確認する確認情報を、マネジメントサーバ１８０２に対して送信する。家庭用ＰＣ７０２についてのレジストリにおける適切な情報がこの後更新される。連絡情報についての変更情報を有するデータパッケージがこの後格納サーバに格納されていなければ、データパッケージは家庭用ＰＣ７０２にダウンロードされない。

【０１４０】様々なクラスのデータについて変更がなされるにつれて、データパッケージは、格納サーバ３００$_1$、３００$_2$に蓄積し、格納空間を費やしてしまう。格納されたデータパッケージの数が増加するにつれて、格納サーバで利用できる格納空間が減少していく。上述した例では、データパッケージCONT.D000は２キロバイト（Ｋ）を占有し、CONT.D001は１Ｋを占有し、CONT.D002は０．５Ｋを占有する。この結果、格納サーバで合計３．５Ｋの格納空間がこれらの３つのファイルにより占有される。格納空間が例えば２５メガバイト（Ｍ）に限定され、かつ、ユーザアカウントに制限が加えられることがあるような状況では、格納サーバにユーザが利用可能な格納空間がなくなるまで、利用可能な格納空間の量は情報が更新されるにつれて減少し続ける。この後、ユーザは、もう変更ログを格納することができないので、ストレスを感じ、データ転送および同期システム全体に対して不満を感じるであろう。

【０１４１】当業者であってもなくても、次に示すようなシナリオにおけるユーザのフラストレーションを理解できるであろう。この例では、ユーザは自分の家庭用ＰＣ７０２におけるMicrosoft社のOutlookの「インボックス」に２０００通の電子メールを持っている。このユーザは、オフィス用ＰＣ７１０のようなその他の自分の装置を家庭用ＰＣ７０２に同期させたいと思う。よって、データパッケージMAIL.D000が、家庭用ＰＣ７０２におけるクライアントソフトウェアにより生成され、かつ、格納を目的としてネットワーク７００にアップロードされる。このデータパッケージは、２０００通すべての電子メールを含んでおり、それぞれのデータパッケージは、関連するＩＤ番号および関連するアクション「追加」を有している。この例では、データパッケージMAIL.D000は、自分のアカウントに割り当てられた全２５Ｍを有するユーザについて、ほぼ１０Ｍのメモリを占有する。これを認識して、このユーザは、占有された空間の量を減らすために、２０００通のメッセージのうちの１５００通を消去する消去コマンドを実行する。この後、１５００の「消去」アクションを含んだ、新データパッケージMAIL.D001が生成される。これらのアクションのそれぞれは、データパッケージMAIL.D000における電子メールＩＤ番号のアクションのうちの特定のものに関連している。新データパッケージMAIL.D001は、さらに１Ｍのメモリを占有するので、結果としてサーバの格納空間がさらに占有されてしまう。この結果、すべてのイベントにおけるユーザは、占有された格納空間の量を１０Ｍからこの値の約１／４すなわち２．５Ｍにまで減らすことを期待するけれども、実際にはその量は１１Ｍに増加する。

46

【０１４２】したがって、可能であれば常に、格納サーバに格納されているデータパッケージを、可能であれば常に崩壊させることが望ましい。本発明の一実施形態に基づいて提供されるように、データパッケージを崩壊させる際には、通常、特定クラスのデータのためのデータパッケージを、消去されている余計な情報に結合させる必要がある。上述した例を用いると、データパッケージMAIL.D000とMAIL.D001は、新データパッケージMAIL.D001を明確にするために、MAIL.D001における「消去」アクションが前データパッケージMAIL.D000における同一のＩＤ番号用の「追加」に置き換わるように、結合する。別の例では、「消去」コマンドは、所定トランザクションにおける１つ以上のフィールドにしか適用されない。この新データパッケージは、好ましくは、オリジナルのMAIL.D001、すなわち「消去」アクションのみを有するものの上に上書きされる。したがって、データパッケージMAIL.D000はもはや関係のないものとなるので消去される。この方法による「ベースをロール（コピー）すること」を実行した後、格納サーバには新MAIL.D001データパッケージのみが残り、この結果、占有された格納空間の量は、ユーザが期待していたように、１０Ｍから約２．５Ｍにまで減少する。

【０１４３】図１８において、本発明の一実施形態に基づいて構成されたベースローリングエンジン１８０６は、データパッケージを期待通りに崩壊させるように設けられている。ベースローリングエンジン１８０６は、望ましくは、ネットワーク７００に接続された装置エンジン１８０８の１つの中に配置される。一実施形態では、ベースローリングエンジン１８０６の実行はユーザにより制御されるが、別の実施形態では、ベースローリングエンジンは、データ転送および同期システムにより自動的に実行される。ベースローリングエンジンが手動で実行される場合には、ユーザは、家庭用ＰＣ７０２のような自分の装置の１つにより自分のアカウントにアク

セスし、自分のデータを圧縮するためのコマンドを出す。一例としては、このコマンドは、ユーザのディスプレイスクリーンにグラフィカルインターフェイスの一部として表示されている「ベースをロールする」ボタンの上にポインタを単に移動させ、マウスまたはトラックボールのような制御装置を用いてこのボタンをクリックすることにより、実行される。

【０１４４】図１９は、本発明の一実施形態に基づいて実行される、データパッケージの崩壊方法を示す図である。上述したように、各データパッケージすなわち各変更ログは、格納サーバ３００$_1$、３００$_2$のようなレポジトリに格納されている。各変更ログ内には、複数のアイテム、一例としては、親ＩＤ、ＩＤ、アクション、および１つ以上のフィールドが含まれている。親ＩＤは、例えば、アイテムが階層的に関連している状況において、特定アイテムと別のアイテムとの関係を識別する。ＩＤおよびアクションアイテムは、上述したように定義されている。各データパッケージのフィールドは、どの特定フィールド、例えばデータのクラスを変更すべきなのかを識別する。各フィールドは、好ましくは、固有の数値を有し、かつ、フィールドについての変更情報を表現する属性を含む。

【０１４５】図１９には、ベースバージョンＤ０００およびこの後のバージョンＤ００１の２つのデータパッケージが存在している。両バージョンとも、上述したアイテムと、１つ以上のフィールドを含む。具体的には、ベースバージョンＤ０００は、ＩＤとしての２と、アクション「追加」と、３つのフィールド、すなわち、属性"John"を持った「ファーストネーム」、属性"Smith"を持った「ラストネーム」、関連するＵＲＬ"http://..."を持った「ウェブページ」とを有する。バージョンＤ００２は、ＩＤとして２を持っているが、変更アクションは「追加」ではなく「変更」である。バージョンＤ００１は、１つのフィールド、すなわち、属性"Scott"を持った「ファーストネーム」のみしか有していない。

【０１４６】図１９には、ユーザが「ベースをロールする」ためのコマンドを出すと、ベースローリングエンジンは、バージョンＤ０００およびＤ００２が１つ以上の同一のＩＤ番号を持っているかどうかを決定する。この例では、Ｄ０００もＤ００２も同一のＩＤ番号２を有しているので、２つのデータパッケージは崩壊して１つのファイルになる。具体的には、新バージョンＤ００２は、生成されて、オリジナルのＤ００２データパッケージに置き換わる。親ＩＤおよびＩＤ番号の２は変わらずに、Ｄ００２からの変更情報はバージョンＤ０００に適用される。具体的には、Ｄ０００とＤ００２の両方が共通して持っているフィールド「ファーストネーム」は、識別されて、データパッケージＤ００２からの最新の属性"Scott"が割り当てられる。アクションは「追加」のままであり、フィールド「ラストネーム」および「ウェブページ」は、それぞれバージョンＤ０００からの"Smith"および"ＵＲＬ"http://..."のままである。これにより、この例では、新しいＤ００２は、フィールド「ファーストネーム」が属性"Scott"に置き換わったことを除いて、Ｄ０００と本質的に同一である。元々のデータパッケージＤ００２に含まれた変更事項がデータパッケージＤ０００にも実際になされ、この結果、新しいデータパッケージＤ０００２が生成される。この後、データパッケージＤ０００は消去される。

【０１４７】図２０に示す別の実施形態では、３つの装置、すなわち、装置Ａ、装置Ｂおよび装置Ｃは、データネットワーク７００に接続することが可能である。一例では、装置Ａはパームトップコンピュータであり、装置Ｂは家庭用ＰＣであり、装置Ｃはオフィス用ＰＣである。別の例では、上述したような様々な別の装置が、装置Ａ、ＢおよびＣとして用いられる。図２０には、連絡情報について少なくとも１５個の異なる変更ログがデータネットワークにアップロードされているが、装置Ａ、ＢおよびＣは、それぞれ異なるバージョンの連絡情報を有する。装置Ａの連絡情報は、ＣＯＮＴ．Ｄ００３における変更事項を含むようにしか更新されておらず、装置ＢはＣＯＮＴ．Ｄ０１０を組み込むように更新され、装置ＣはＣＯＮＴ．Ｄ０１５に更新されている。この例では、ＤＯＸＸは連絡情報の連続的なバージョンを表す。すなわち、このシーケンスにおける１番目の変更ログはＣＯＮＴ．Ｄ０００であり、このシーケンスにおける１５番目の変更ログはＣＯＮＴ．Ｄ０１５である。これらの変更ログすべては、データネットワークに接続された格納サーバに格納されている。

【０１４８】図２０では、様々な装置間における連絡情報のバージョンは、分割されている、すなわち、連続していないが、格納サーバに格納された１５のデータパッケージは、本発明の一実施形態に基づいて崩壊される。第１に、複数のベースバージョンが定められる。この例では、第１ベースデータパッケージは、ＣＯＮＴ．Ｄ０００から始まって装置Ａのバージョン（この例ではＣＯＮＴ．Ｄ００３）まで続く上述したような連続的なデータパッケージを崩壊させることにより、定められる。第２ベースデータパッケージは、装置ＡのバージョンＣＯＮＴ．Ｄ００３および崩壊している連続した変更ログから、装置Ｂのバージョン（この例ではＣＯＮＴ．Ｄ０１０）までにわたって、定められる。第３ベースデータパッケージは、ＣＯＮＴ．Ｄ０１０とＣＯＮＴ．Ｄ０１５との間における、崩壊している連続したデータパッケージにより、同様に定められる。これにより、３つの連続したベースデータパッケージが、それぞれ、データパッケージＣＯＮＴ．Ｄ０１３、Ｄ０１４およびＤ０１５に置き換わる。この後、装置Ａは新しいＣＯＮＴ．Ｄ０１３を含む連絡情報に対する変更事項を含むように、装置ＢはＣＯＮＴ．Ｄ０１４を含むように、装置Ｃはバージ

ョンＤ０１５を含むように、それぞれ更新される。

【０１４９】新ベースデータパッケージを定める処理を通して、マネジメントサーバ１８０２は、各装置についてのレジストリにおいて、その装置に格納された最新バージョン番号の連絡情報を維持する。上記した例では、崩壊動作の前では、装置Ａはバージョン３の状態であり、装置Ｂはバージョン１０の状態であり、装置Ｃはバージョン１５の状態である。ベースローリングエンジンはマネジメントサーバ１８０２と通信を行っているので、データパッケージが崩壊する際には、ベースローリングエンジンは、マネジメントサーバ１８０２に対して特定ユーザについての装置情報を要求して、このマネジメントサーバ１８０２から受信する。この装置情報は、ユーザによって登録された装置すべてを識別し、かつ、各装置に格納されたデータクラスの最新バージョンがどのようなものであるかを識別する情報を含んでいる。

【０１５０】この結果、崩壊動作の後に、装置Ｂを同期させる際には、マネジメントサーバは装置Ｂがバージョン１４の状態にあることを認識するので、データパッケージＣＯＮＴ．Ｄ０１５のみを装置Ｂにダウンロードする必要がある。同様に、装置Ａが同期を行うためにネットワークに接続する際には、ＣＯＮＴ．Ｄ０１４における変更情報が最初に適用され、この後、ＣＯＮＴ．Ｄ０１５における変更情報が適用される。このような更新はすべて上述した技術を用いて実現される。

【０１５１】装置がネットワークに接続される間に、１つか２つの更新を実行することは、計算的に複雑ではないので、更新の全シーケンスを実行することよりも速くなされる。これは、すべてのファイルをダウンロードするためには、通信チャネルを確立し、ファイルをダウンロードし、この後、このチャネルを閉じる必要がある、という事実に基づく。このように接続を開閉することに伴って高オーバーヘッドが生ずる。これを多く繰り返すにつれて、より多くの時間がかかるので、より高いコストがかかってしまう。上記例では、装置Ａにバージョン番号１３の現行情報を与えるためにベースを上述したようにロールしないとすれば、装置Ａをバージョン番号３からバージョン番号１５に更新することが必要となるであろう。上記例を用いると、１２個のファイルについては、接続を開き、ファイルをダウンロードし、接続を閉じることを、１２回行わなければならない。３個のファイルについては、このような繰り返しを３回だけ実行する必要がある。より少ないデータパッケージがネットワークから装置Ａに対して送信されるので、データの量が小さくなる。この結果、徐々に、装置Ａによる処理が少なくなり、効率が向上する。

【０１５２】本発明の上述した一実施形態によれば、ネットワークサービスを介して個人情報を届けるためのユーザ中心の通信モデルを提供することができる。このモデルは、様々な時間に、インターネットのようなネットワークに接続されていない装置を収容することができる。存在する情報にサーバ中心のモデルを押し付けるのではなく、個人情報は局部的に存在し続けることができる。

【０１５３】上述した説明によれば、格納および転送情報のブロードキャストが用いられる。存在する情報に対する変更事項は、インターネット格納サーバにコピーが取られた後、変更事項は、ネットワークにおける他の装置によって装置に特化した時間に検索される。この方法により、１対１の通信を必要とすることなく、直接クライアント通信（direct client communication）が実現される。１つの通信が本発明のシステムによってサポートされているが、これは必要ではない。

【０１５４】本発明は、様々なタイプの装置の間で個人情報を同期させることを目的として、インターネット格納および転送ブロードキャストの形で表現されているが、上述したシステムのためのアプリケーション専用として、同期を実現する必要はないことは、用意に理解できる。具体的には、システムは、一部分のデータのみしかクライアントアプリケーションで変更する必要がないようないわゆる「プッシュ」がた情報アプリケーションにおける情報に対する変更事項を効率的にブロードキャストするために用いられうる。例えば、ストックプライスの変更事項のような情報を複数のユーザに対してブロードキャストする必要があるようなシステムでは、上述した技術を実行するクライアントアプリケーションは、特定のストックプライスに関連した、クライアントアプリケーションにおける特定部分のデータを変更することのみにより、更新される。これは、他の装置を用いて以前に決定されているものよりも小さいバンド幅を用いて実行されうる。

【０１５５】上述した一実施形態が本発明の原理を説明するだけのためものであることを理解されたい。さらなる変更は当業者にとって明らかであるので、本発明の範囲および思想から逸脱することなくこのような変更を施すことが可能である。よって、本発明は、上述した特定の詳細事項に限定されない。むしろ、別記請求項が本発明の範囲内および思想内にあるような変形および変更を含む、ということが意図されている。

【図面の簡単な説明】

【図１】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図２】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図３】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図４】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図５】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

51

【図６】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図７】本発明の一実施形態に基づいて構成されたデータ転送および同期システムの一般化されたブロック図

【図８】本発明の一実施形態に基づいて構成されたデータ転送および同期システムにおけるシステム構造の一般化されたブロック図

【図９Ａ】本発明の一実施形態に基づいて構成されたデスクトップ装置エンジンの一般化されたブロック図

【図９Ｂ】本発明の一実施形態に基づいて構成されたサーバ側装置エンジンの一般化されたブロック図

【図１０】本発明の一実施形態に基づいて、Windows（登録商標）のようなオペレーティングシステム環境におけるデスクトップ装置エンジンの一般化されたブロック図

【図１１】本発明の一実施形態に基づいて構成された装置エンジンに組み込まれるアプリケーションオブジェクトの一般化されたブロック図

【図１２】本発明の一実施形態に基づいて構成されたシステムに用いられる汎用データフォーマットの格納オブジェクト階層の一般化されたブロック図

52

【図１３】本発明の一実施形態に基づいて実行されるルーチンに用いるアイテムオブジェクトのリスト

【図１４】本発明の一実施形態に基づいて構成されるマネジメント格納サーバの一般化されたブロック図

【図１５】本発明の一実施形態に基づいて実行されるプル同期を示す一般化されたフロー図

【図１６】本発明の一実施形態に基づいて実行されるプッシュ同期を示す一般化されたフロー図

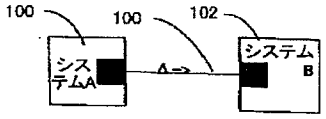
【図１７】本発明の一実施形態に基づいて構成されたマネジメントサーバ構造の一般化されたブロック図

【図１８】本発明の一実施形態に基づいて構成されたベースロールエンジンを有するデータ転送および同期システムの一般化されたブロック図

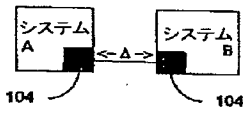
【図１９】本発明の一実施形態に基づいて実行されるデータパッケージを崩壊させる様子を示す図

【図２０】それぞれが本発明の一実施形態に基づいて実行される異なるバージョンのアプリケーション情報を有する、データネットワークに接続された複数の装置のためのデータパッケージを崩壊させる様子を示す図

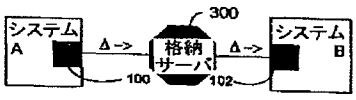
【図１】



【図２】

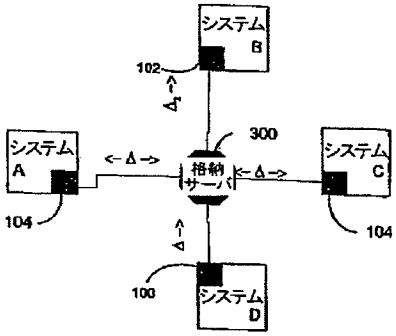


【図３】

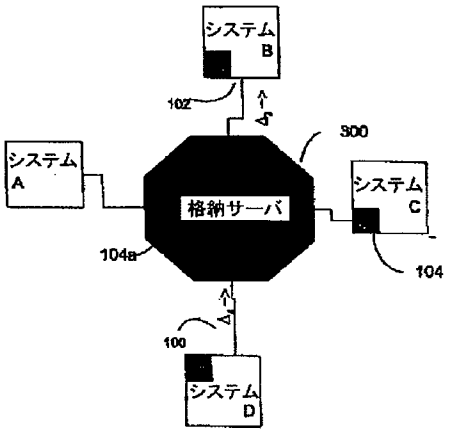


【図４】

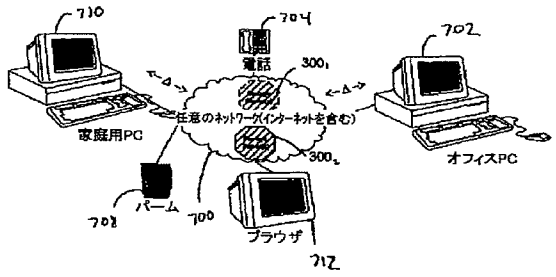
【図５】



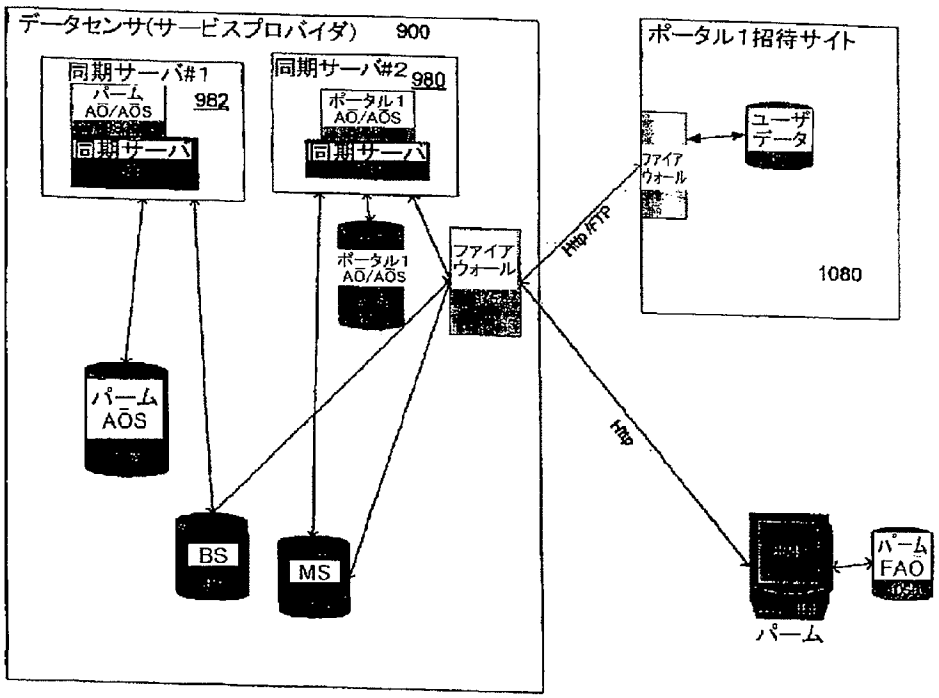
【図６】


システム
B
102
300
システム
A
格納サーバ
システム
C
104a
104
100
システム
D

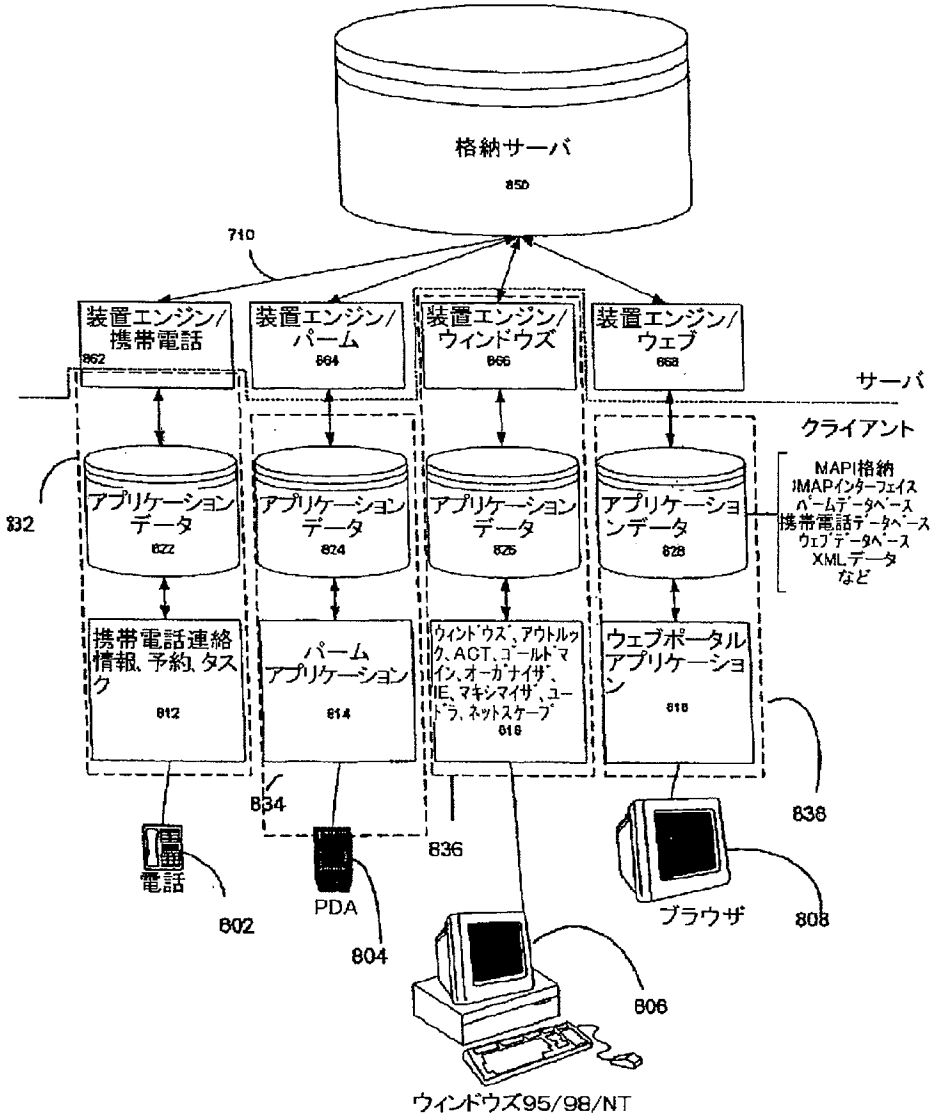
【図７】


710
704
電話
702
300₁
任意のネットワーク(インターネットを含む)
家庭用PC
300₂
オフィスPC
パーム
708
700
ブラウザ
712

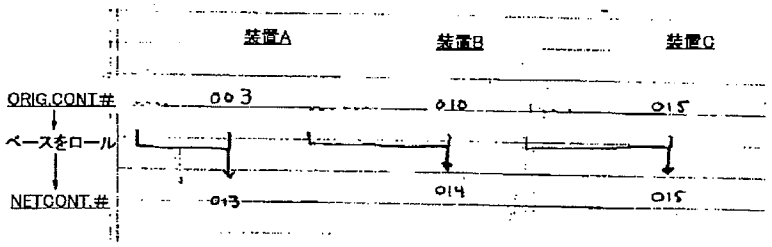
【図９Ｂ】


データセンサ(サービスプロバイダ)　900
同期サーバ#1
パーム
AO/AOS
982
同期サーバ
同期サーバ#2
980
ポータル1
AO/AOS
同期サーバ
ポータル1
AO/AOS
ファイア
ウォール
パーム
AOS
BS
MS
ポータル1招待サイト
ファイア
ウォール
ユーザ
データ
1080
http/FTP
http
パーム
FAO
パーム

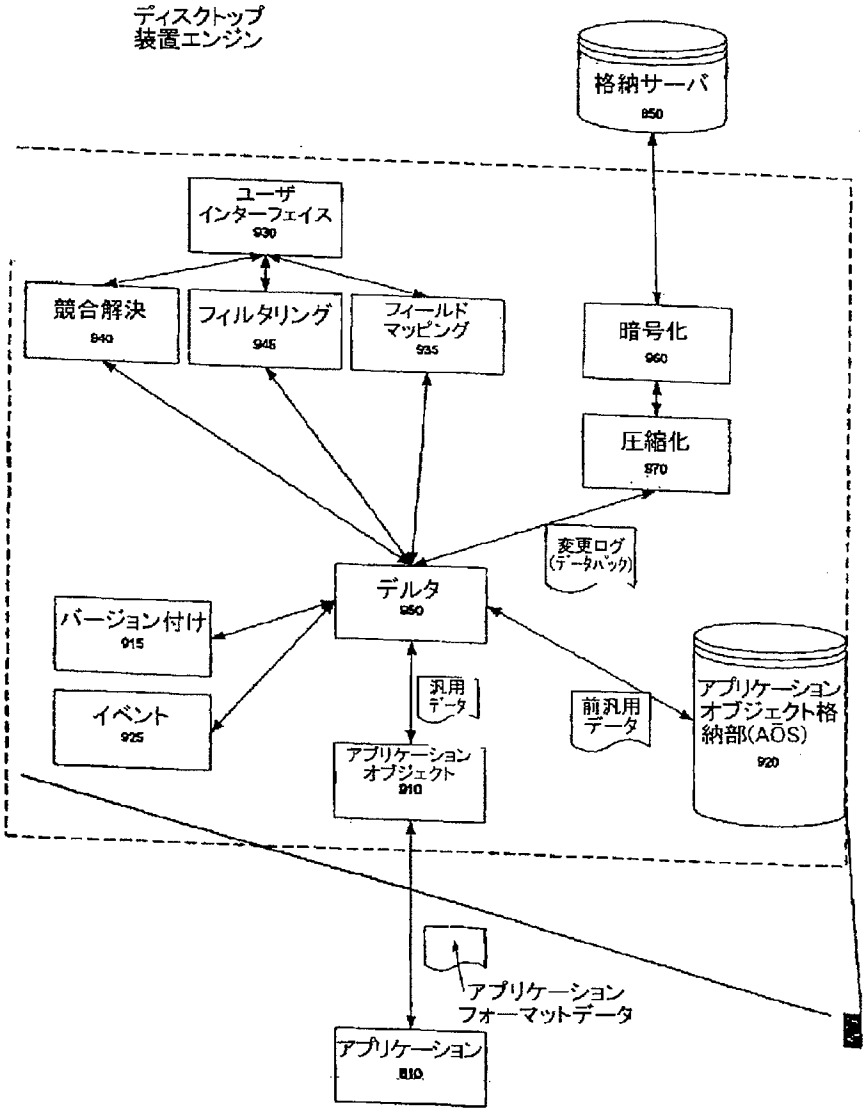
【図８】


格納サーバ
850
710
装置エンジン/携帯電話
862
装置エンジン/パーム
864
装置エンジン/ウィンドウズ
866
装置エンジン/ウェブ
868
サーバ
クライアント
832
アプリケーションデータ
822
アプリケーションデータ
824
アプリケーションデータ
826
アプリケーションデータ
828
MAPI格納
IMAPインターフェイス
パームデータベース
携帯電話データベース
ウェブデータベース
XMLデータ
など
携帯電話連絡情報、予約、タスク
812
パームアプリケーション
814
ウィンドウズ、アウトルック、ACT、ゴールドマイン、オーガナイザ、IE、マキシマイザ、ユードラ、ネットスケープ
816
ウェブポータルアプリケーション
818
834
836
838
電話
802
PDA
804
ブラウザ
808
806
ウィンドウズ95/98/NT

【図２０】

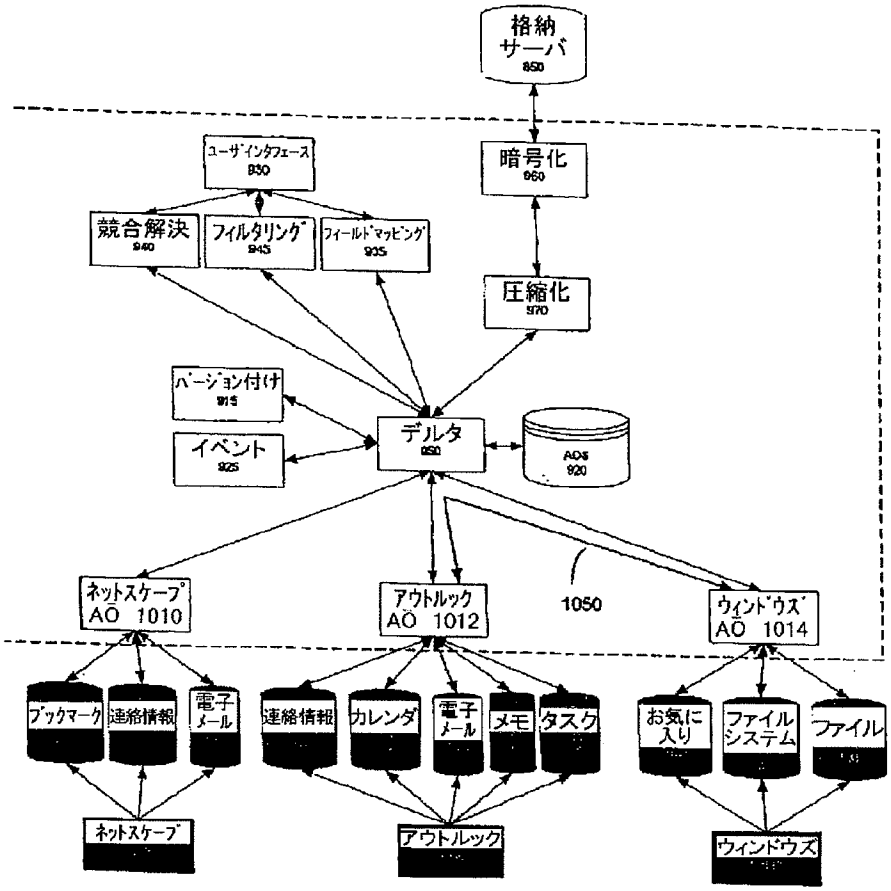

装置A
装置B
装置C
ORIG.CONT#
003
010
015
ベースをロール
NETCONT#
013
014
015

【図９Ａ】

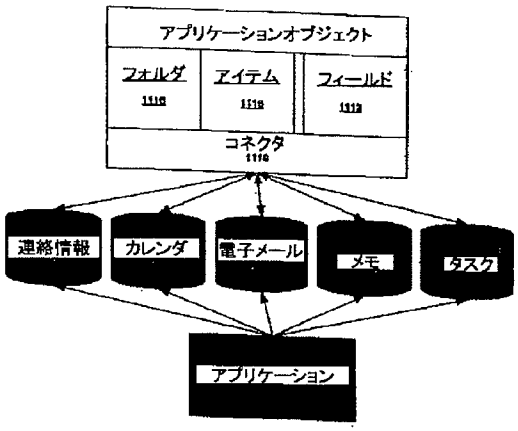

ディスクトップ
装置エンジン
格納サーバ
850
ユーザ
インターフェイス
930
競合解決
940
フィルタリング
945
フィールド
マッピング
935
暗号化
960
圧縮化
970
変更ログ
(データパック)
デルタ
950
バージョン付け
915
イベント
925
汎用
データ
アプリケーション
オブジェクト
910
前汎用
データ
アプリケーション
オブジェクト格
納部(AOS)
920
アプリケーション
フォーマットデータ
アプリケーション
810

【図１０】

装置エンジン/ウィンドウズ

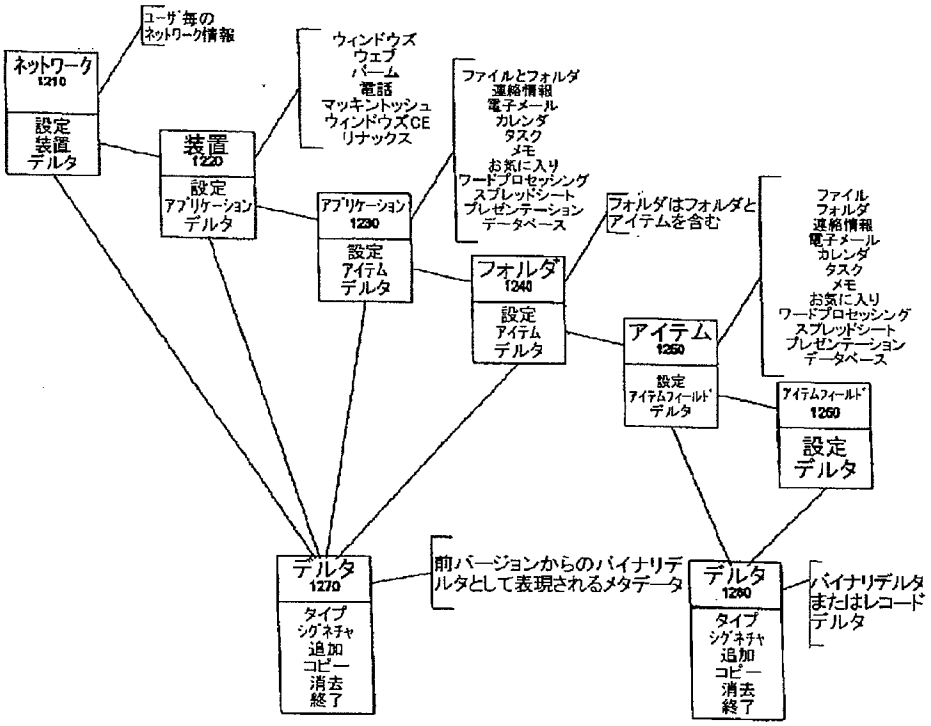


【図１１】

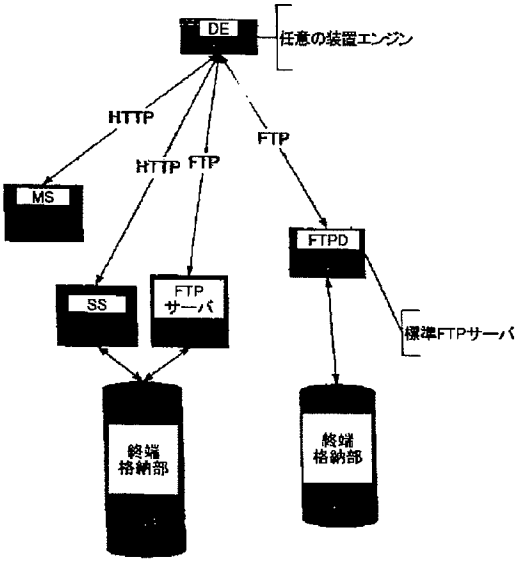


【図１２】

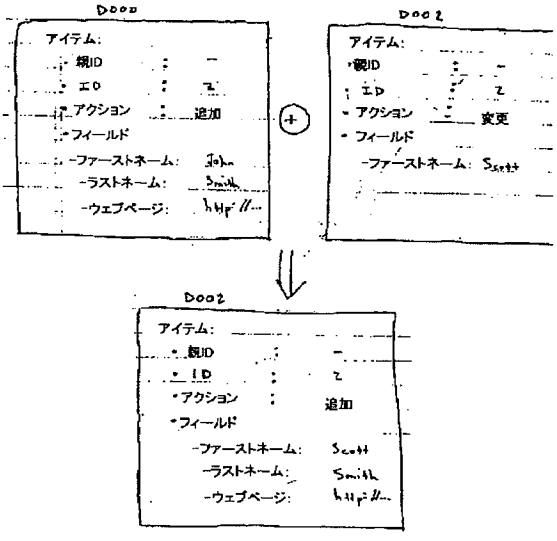
オブジェクト階層



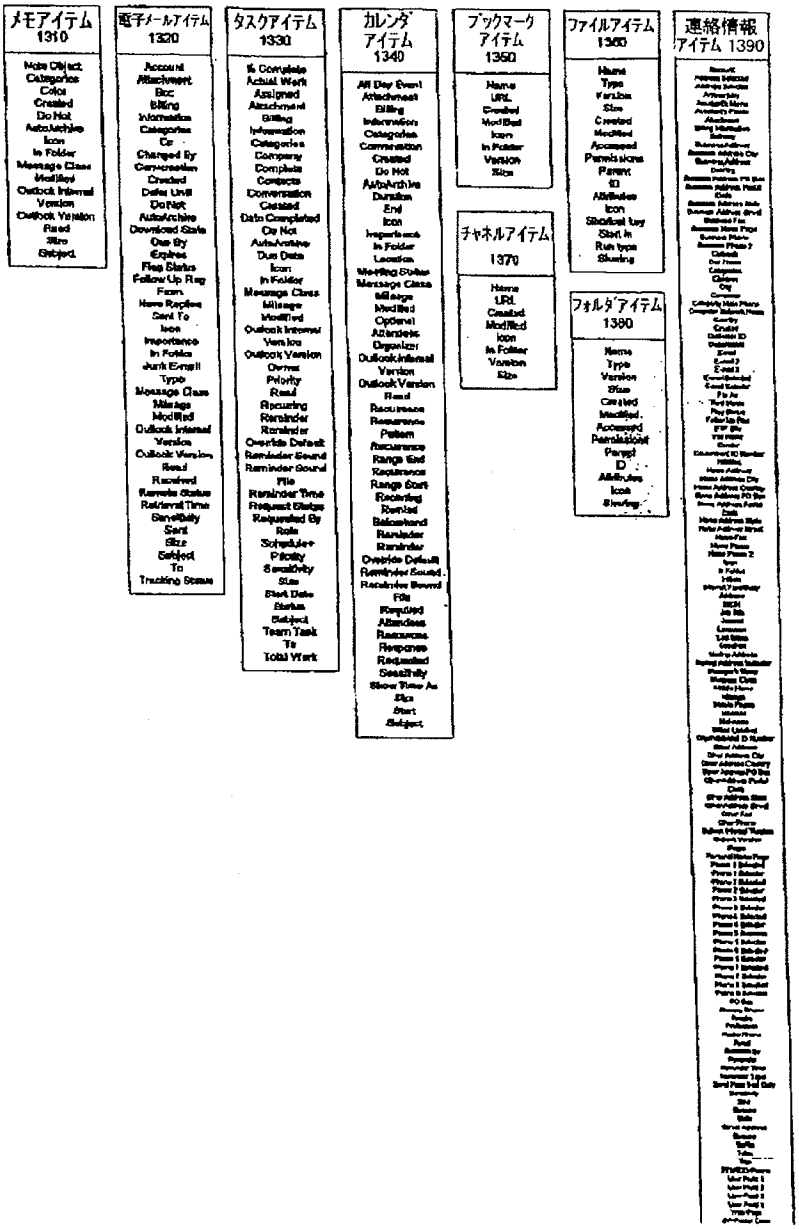
【図１４】



【図１９】

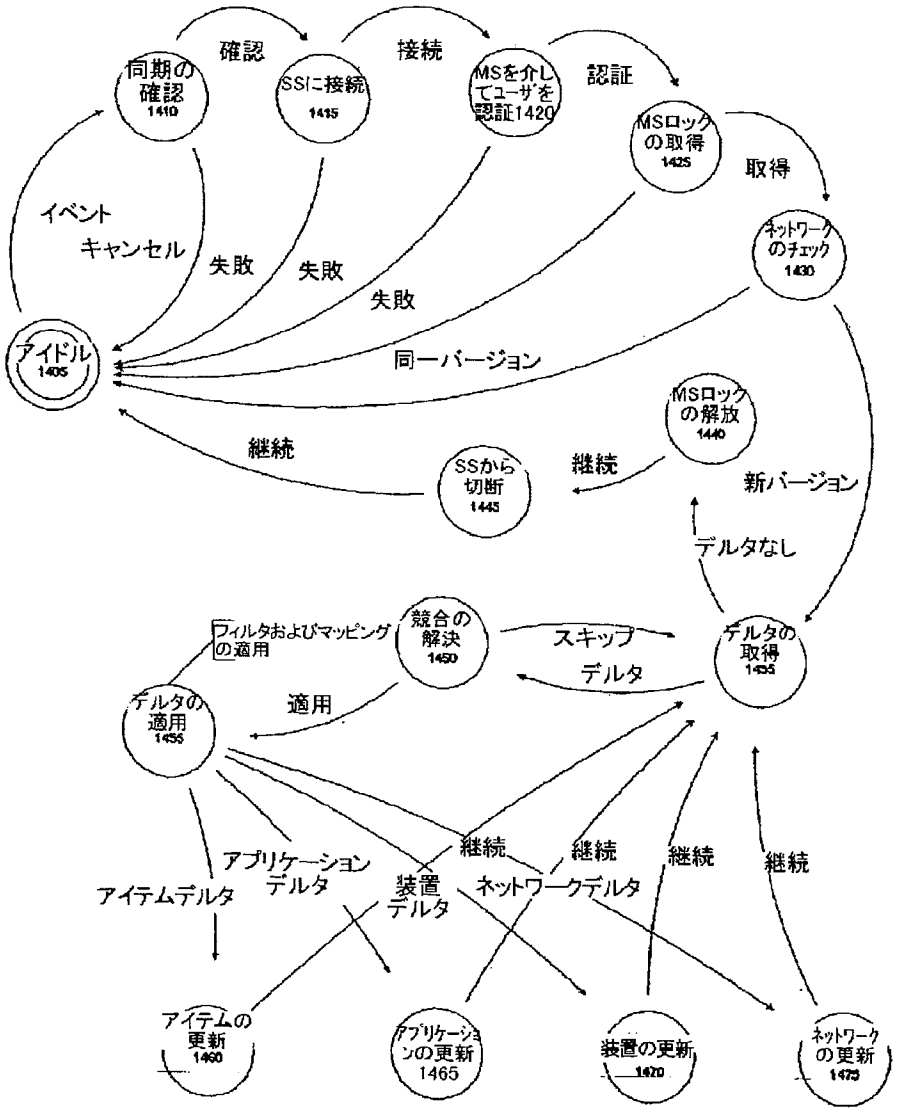


【図１３】


メモアイテム
1310
Note Object
Categories
Color
Created
Do Not
AutoArchive
Icon
In Folder
Message Class
Modified
Outlook Internal
Version
Outlook Version
Read
Size
Subject
電子メールアイテム
1320
Account
Attachment
Bcc
Billing
Information
Categories
Cc
Changed By
Conversation
Created
Defer Until
Do Not
AutoArchive
Download State
Due By
Expires
Flag Status
Follow Up Flag
From
Have Replies
Sent To
Icon
Importance
In Folder
Junk Email
Type
Message Class
Mileage
Modified
Outlook Internal
Version
Outlook Version
Read
Received
Remote Status
Retrieval Time
Sensitivity
Sent
Size
Subject
To
Tracking Status
タスクアイテム
1330
% Complete
Actual Work
Assigned
Attachment
Billing
Information
Categories
Company
Complete
Contacts
Conversation
Created
Date Completed
Do Not
AutoArchive
Due Date
Icon
In Folder
Message Class
Mileage
Modified
Outlook Internal
Version
Outlook Version
Owner
Priority
Read
Recurring
Reminder
Reminder
Override Default
Reminder Sound
Reminder Sound
File
Reminder Time
Request Status
Requested By
Role
Schedule+
Priority
Sensitivity
Size
Start Date
Status
Subject
Team Task
To
Total Work
カレンダアイテム
1340
All Day Event
Attachment
Billing
Information
Categories
Conversation
Created
Do Not
AutoArchive
Duration
End
Icon
Importance
In Folder
Location
Meeting Status
Message Class
Mileage
Modified
Optional
Attendees
Organizer
Outlook Internal
Version
Outlook Version
Read
Recurrence
Recurrence
Pattern
Recurrence
Range End
Recurrence
Range Start
Recurring
Remind
Before Hand
Reminder
Reminder
Override Default
Reminder Sound
Reminder Sound
File
Required
Attendees
Resources
Response
Requested
Sensitivity
Show Time As
Size
Start
Subject
ブックマークアイテム
1350
Name
URL
Created
Modified
Icon
In Folder
Version
Size
チャネルアイテム
1370
Name
URL
Created
Modified
Icon
In Folder
Version
Size
ファイルアイテム
1360
Name
Type
Version
Size
Created
Modified
Accessed
Permissions
Parent
ID
Attributes
Icon
Shortcut key
Start In
Run type
Sharing
フォルダアイテム
1380
Name
Type
Version
Size
Created
Modified
Accessed
Permissions
Parent
ID
Attributes
Icon
Sharing
連絡情報
アイテム 1390

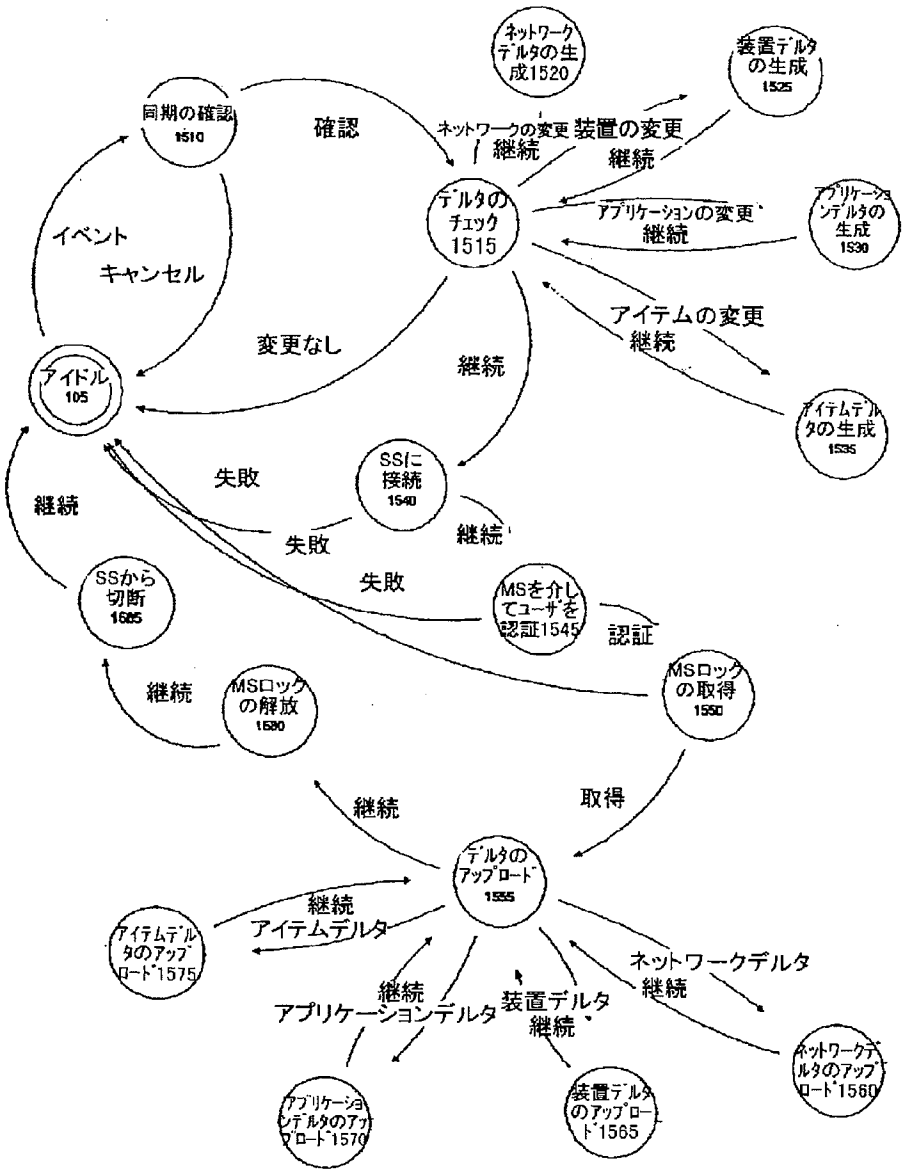
【図１５】

プル同期

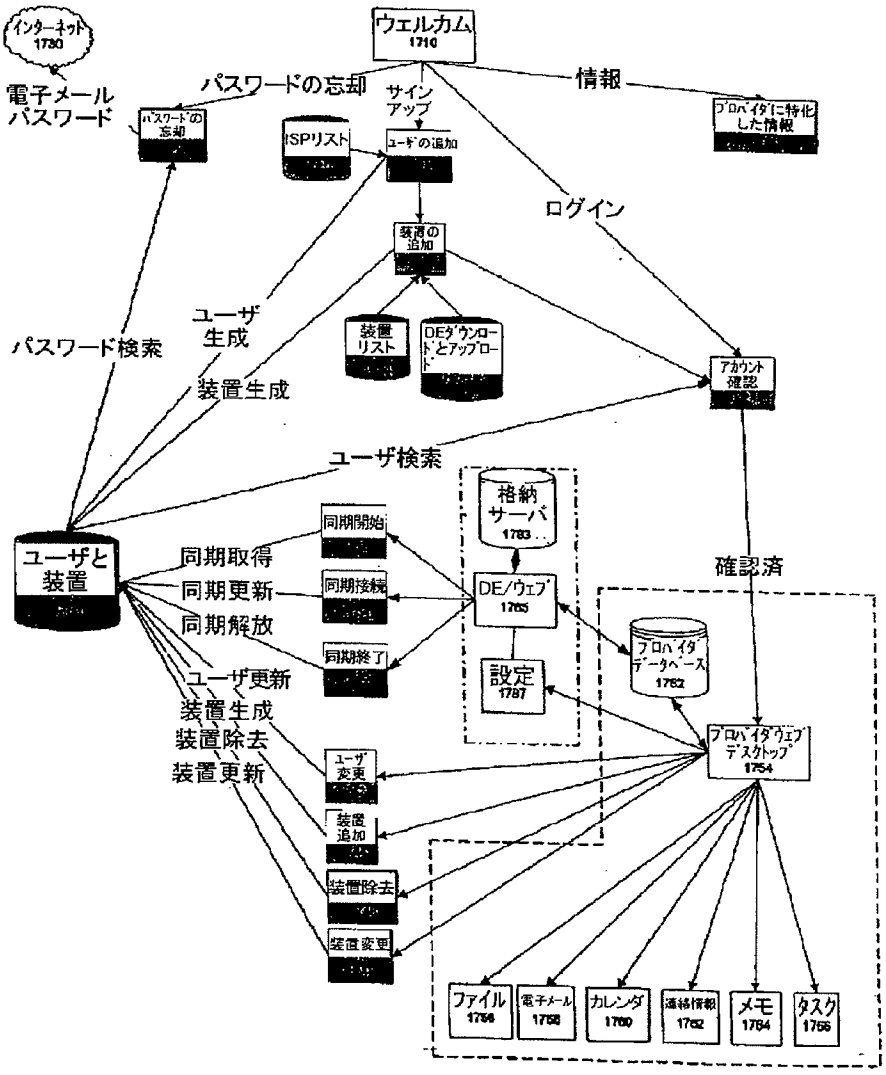

同期の確認 1410
確認
SSに接続 1415
接続
MSを介してユーザを認証1420
認証
MSロックの取得 1425
取得
ネットワークのチェック 1430
イベント
キャンセル
失敗
失敗
失敗
アイドル 1405
同一バージョン
MSロックの解放 1440
継続
SSから切断 1445
継続
新バージョン
デルタなし
フィルタおよびマッピングの適用
競合の解決 1450
スキップ
デルタ
デルタの取得 1455
デルタの適用 1455
適用
継続
継続
継続
継続
アイテムデルタ
アプリケーションデルタ
装置デルタ
ネットワークデルタ
アイテムの更新 1460
アプリケーションの更新 1465
装置の更新 1470
ネットワークの更新 1475

【図１６】

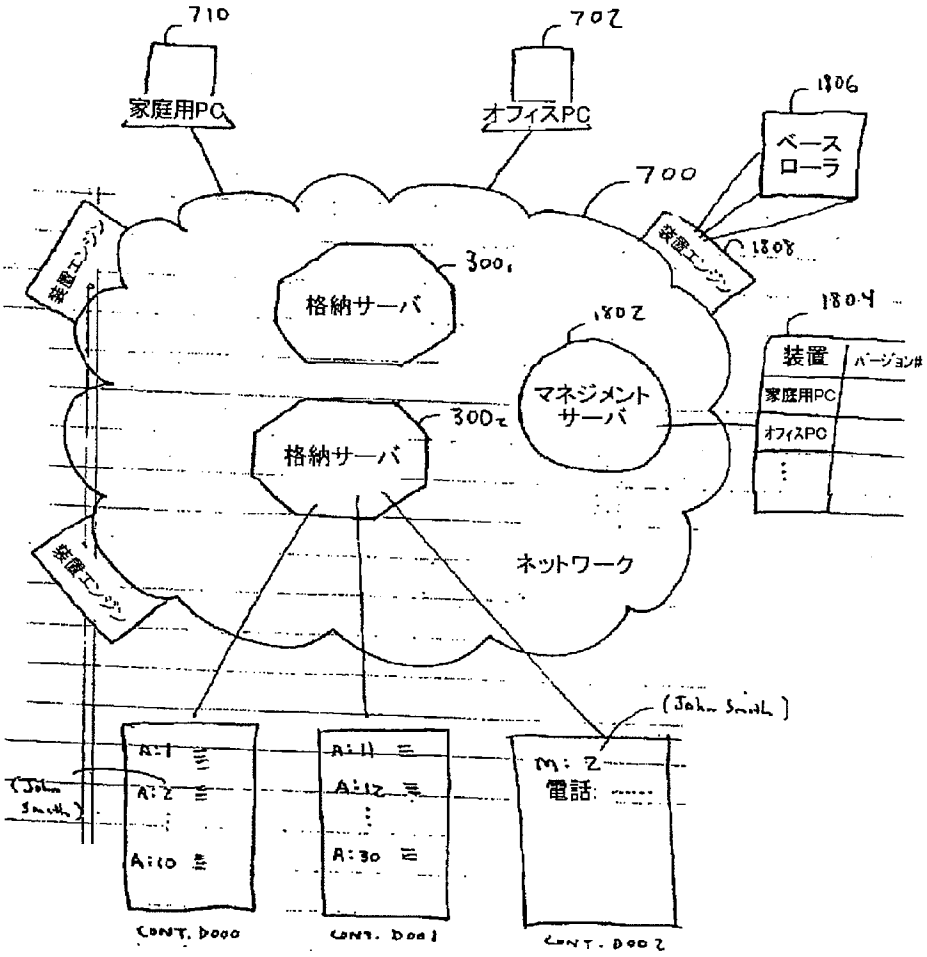
プッシュ同期


ネットワークデルタの生成1520
装置デルタの生成1525
同期の確認1510
確認
ネットワークの変更継続
装置の変更継続
デルタのチェック1515
アプリケーションの変更継続
アプリケーションデルタの生成1530
イベント
キャンセル
アイドル105
変更なし
継続
アイテムの変更継続
アイテムデルタの生成1535
失敗
SSに接続1540
継続
失敗
継続
失敗
SSから切断1685
MSを介してユーザを認証1545
認証
継続
MSロックの解放1580
MSロックの取得1550
継続
取得
デルタのアップロード1555
継続
アイテムデルタ
アイテムデルタのアップロード1575
継続
アプリケーションデルタ
装置デルタ継続
ネットワークデルタ継続
ネットワークデルタのアップロード1560
アプリケーションデルタのアップロード1570
装置デルタのアップロード1565

【図１７】


インターネット
1750
電子メール
パスワード
ウェルカム
1710
パスワードの忘却
サイン
アップ
情報
パスワードの
忘却
ISPリスト
ユーザの追加
プロバイダに特化
した情報
ログイン
装置の
追加
パスワード検索
ユーザ
生成
装置
リスト
DEダウンロード
とアップロード
装置生成
アカウント
確認
ユーザ検索
格納
サーバ
1783
同期開始
ユーザと
装置
同期取得
同期更新
同期解放
同期接続
同期終了
DE/ウェブ
1765
確認済
プロバイダ
データベース
1762
設定
1787
ユーザ更新
装置生成
装置除去
装置更新
プロバイダウェブ
デスクトップ
1754
ユーザ
変更
装置
追加
装置除去
装置変更
ファイル
1756
電子メール
1758
カレンダ
1760
連絡情報
1762
メモ
1764
タスク
1766

【図１８】



フロントページの続き

(72)発明者　ロバート　イー　ガーナー
アメリカ合衆国　ジョージア州　30043
ローレンスヴィル　ヒドゥン　ウッド　コート　309

(72)発明者　レイトン　エイ　リッドガード
アメリカ合衆国　ジョージア州　30294
エレンウッド　フレイクス　ミル　マナー　レーン　4152

(72)発明者　ライアム　ジェイ　スタナード
アメリカ合衆国　ジョージア州　30043
ローレンスヴィル　プロスペクト　ロード　1584

(72)発明者　ドナルド　ダブリュ　キャッシュ
アメリカ合衆国　ジョージア州　30338
ダンウッディー　ヴァンダーリン　ドライヴ　1748

(72)発明者　ジョセフ　ロバートソン
アメリカ合衆国　ジョージア州　30683
ウィンターヴィル　チャーリー　ボルトン　ロード　802

Ｆターム(参考)　5B082 GB02 HA05

【外国語明細書】

BASE ROLLING ENGINE FOR DATA TRANSFER AND SYNCHRONIZATION SYSTEM

REFERENCE TO EARLIER FILED APPLICATIONS

This is a continuation-in-part of the following co-pending applications: U.S. Application No. 09/490,550, filed January 25, 2000, U.S. Application No. 09/491,675 and U.S. Application No. 09/491,694, filed January 26, 2000, all entitled "Data Transfer and Synchronization System."

COPYRIGHT NOTICE

FIELD

The invention relates to the transference of data between two systems independent of the form in which the data is kept on the respective systems, and in particular to providing an efficient means of communicating data between systems and devices.

BACKGROUND

The growth of computing-related devices has not been limited to personal computers or work stations. The number of personal computing devices has grown

substantially in both type and format. Small, hand-held computers carry a multitude of contact, personal, document, and other information and are sophisticated enough to allow a user to fax, send e-mails, and communicate in other ways wirelessly. Even advanced cellular phones carry enough memory and processing power to store contact information, surf the web, and provide text messaging. Along with the growth in the sophistication of these devices, the need to transfer information between them has grown significantly as well.

With a multitude of different device types on the market, keeping information synchronized among the different devices has become increasingly problematic. For example, an individual keeps a calendar of information on a personal computer in his or her office using a particular personal information manager application. This individual would generally like to have the same information available in a cellular phone, hand-held organizer, and perhaps a home personal computer. The individual may additionally have a notebook computer which requires synchronizing file data such as presentations or working documents between the notebook and the office computer.

Until now, synchronization between both documents and personal information managers has occurred through direct connection between the devices, and generally directly between applications such as a personal information manager in one device and a personal information manager in another device or using an intermediary sync-mapping program. One example of this is the prevalent use of the 3Com Palm® OS-based organizer, such as the 3Com Palm® series of computing devices, which uses its own calendaring system, yet lets users synchronize the data therein with a variety of different personal information manager software packages, such as Symantec's ACT!™, Microsoft's Outlook®, and other systems. In this example, an intermediary synchronization program such as Puma Technology, Inc.'s Intellisync® is required. Intellisync® is an application program which runs on both the hand-held device and the computer which stores the information data and maps data systems between non-uniform data records. In other cases, direct transfer between applications such as transfer between Microsoft's Outlook® computer-based client

and Microsoft's Windows CE "Pocket Outlook" application, is possible. Nevertheless, in both cases, synchronization occurs through direct connection between a personal computer and the personal computing device. While this connection is generally via a cable directly connecting, for example, Palm® device in a cradle to the personal computer, the connection may be wireless as well.

One component of these synchronization systems is that the synchronization process must be able to delineate between when changes are made to specific databases and must make a decision about whether to replace the changed field. Normally, this is measured by a change in one database, and no-change in a second database. In some cases, both databases will have changed between syncs. In this case, the sync operation must determine which of the two changes which has been made is to "win" and replace the other during the sync. Generally, this determinant of whether a conflict exists allows some means for letting the user resolve the conflict.

In a technical sense, synchronization in this manner is generally accomplished by the copying of full records between systems. At some level, a user is generally required to map data fields from one application to another and specify which data fields are assigned to which corresponding field in a different device. Less mapping is required where developers more robustly support various platforms of applications.

In many instances, the data to be synchronized is generally in the form of text data such as records of addresses, contact information, calendar information, notes and other types of contact information. In certain instances, data to be synchronized will be binary format of executable files or word processor-specific documents. In many cases where document synchronization is required, the synchronization routine simply determines whether or not the documents in question have changed, and uses a time-based representation to determine which of the two files is newer, and replaces the older file with the newer file to achieve synchronization, as long as the older of the two files was in fact not changed. This is the model used in the familiar "Briefcase" function in Microsoft Windows-based systems. If both files have

changed, then the synchronization routine presents the option of conflict resolution to the user. Such synchronization schemes are generally relatively inefficient since they require full band-width of the document or binary file to be transferred via the synchronization link. In addition, at some level the synchronization programs require interaction by the user to map certain fields between different programs.

One of the difficulties in providing synchronization between different computing devices is that the applications and platforms are somewhat diverse. Nevertheless, all synchronization programs generally require certain functions in order to be viable for widespread usage. In particular, synchronization programs must work with popular applications on various platforms. Sync applications must allow for conflicts resolution when changes are made to the same information on different devices between syncing events. They must provide synchronization for all types of formats of data, whether it be text data in the form of contacts, e-mails, calendar information, memos or other documents, or binary data in the form of documents or programs in particular types of formats.

In a broader sense, applications which efficiently synchronize data between disparate types of devices can provide advantages in applications beyond synchronizing individual, personal information between, for example, a personal information manager hardware device such as a Palm® computing device, and a personal computer. The same objectives which are prevalent in developing data transfer between personal information management (PIM) devices and desktop systems lend themselves to furthering applications requiring data transfer between other types of devices, on differing platforms. These objectives include speed, low bandwidth, accuracy, and platform independence.

For example, current e-mail systems use a system which is somewhat akin to the synchronization methods used for disparate devices in that an entire message or file is transferred as a whole between different systems. When a user replies to an e-mail, generally the entire text of the original message is returned to the sender, who now has two copies of the e-mail text he/she originally sent out. The same is true if an e-mail attachment is modified and returned. All of the text which is the

same between both systems is essentially duplicated on the originator's system.

SUMMARY

The present invention relates to a base rolling engine for collapsing data packages stored in a data transfer and synchronization system. A first data package is provided. The first data package has a first transaction including an identification number, an action, and a plurality of fields. Each field has an attribute representing change information. A second data package is also provided. The second data package has a second transaction made subsequent to the first transaction. The second transaction has an identification number, an action, and a field with an attribute. The base rolling engine determines whether the identification number of the second transaction corresponds to the identification number of the first transaction. The base rolling engine also determines whether the field of the second transaction corresponds to one of the fields of the first transaction. When the identification numbers of the first and second transactions correspond to one another, and the field of the second transaction corresponds to one of the fields of the first transaction, the first and second data packages are combined. A combined data package is thus defined having a combined transaction with the identification number. The second data package is replaced with the combined data package.

BRIEF DESCRIPTION OF THE FIGURES

The invention will be described with respect to various exemplary embodiments thereof. Other features and advantages of the invention will become apparent with reference to the specification and drawings in which:

Fig. 1-7 are generalized block diagrams of data transfer and synchronization systems constructed in accordance with exemplary embodiments of the present invention;

Fig. 8 is a generalized block diagram of the system architecture of a data transfer and synchronization system constructed in accordance with an exemplary embodiment of the present invention;

Fig. 9A is a generalized block diagram of a desktop device engine constructed in accordance with an exemplary embodiment of the present invention;

Fig. 9B is a generalized block diagram of a server side device engine constructed in accordance with an exemplary embodiment of the present invention;

Fig. 10 is a generalized block diagram of a desktop device engine in an operating system environment such as Windows, according to an exemplary embodiment of the present invention;

Fig. 11 is a generalized block diagram of an application object incorporated into a device engine constructed according to an exemplary embodiment of the present invention;

Fig. 12 is a generalized block diagram of storage object hierarchy of a universal data format used in accordance with a system constructed in accordance with an exemplary embodiment of the present invention.

Fig. 13 is a listing of exemplary item objects used in accordance with the routines performed in accordance with exemplary embodiments of the present invention.

Fig. 14 is a generalized block diagram of a management storage server constructed in accordance with an exemplary embodiment of the present invention;

Fig. 15 is a generalized flow diagram illustrating a pull synchronization performed in accordance with an exemplary embodiment of the present invention;

Fig. 16 is a generalized flow diagram illustrating a push synchronization performed in accordance with an exemplary embodiment of the present invention;

Fig. 17 is a generalized block diagram of a management server architecture constructed in accordance with an exemplary embodiment of the present invention;

Fig. 18 is a generalized block diagram of a data transfer and synchronization system having a base rolling engine constructed in accordance with an exemplary embodiment of the present invention;

Fig. 19 is a diagram illustrating a collapsing of data packages, performed in accordance with an exemplary embodiment of the present invention; and

Fig. 20 is a diagram illustrating a collapsing of data packages for a plurality of

devices coupled to a data network, each device having a different version of application information, performed in accordance with an exemplary embodiment of the present invention.

DETAILED DESCRIPTION

Fig. 1 is a generalized block diagram of a first data transfer and synchronization system constructed in accordance with an exemplary embodiment of the present invention. A first system or device, system A, and a second system or device, system B, are coupled by a communication line 110. It should be readily understood that communication line 110 may be any direct coupling of the two systems allowing data to pass between the systems. For example, in various embodiments, such coupling includes serial ports, parallel ports, Ethernet connections, other types of networks, infrared links, and the like. In various exemplary embodiments, systems A and/or B are personal computers ("PC"), smart telephones, a cellular phones, personal information computing devices, hand-held computers, notebooks, and web browsers. In other exemplary embodiments, systems A and/or B include hardware components of a computer system, and other combinations of hardware including, for example, a processor and memory adapted to receive and provide information to another device. Other exemplary embodiments of Systems A and/or B include software containing such information and residing on a collection or collections of hardware. Examples of such software include applications such as personal information managers, which include contact data and other such information, e-mail systems, and file systems, such as those used by Microsoft Windows NT operating systems, Unix operating systems, Linux operating systems, and other systems capable of storing file types having binary formats which translate to application formats of differing types.

In Fig. 1, System A includes a functional block 100 representing a differencing transmitter. System B includes a functional block 102 representing a differencing receiver. The differencing transmitter 100, upon receipt of a control signal enabling operation of the transmitter, examines a specified data structure of information which

is to be transmitted to system B. Differencing transmitter 100 extracts such information from System A and converts the information extracted into difference information Δ. Difference information Δ comprises only the changes to System B's data which have occurred on System B and instructions for implementing those changes. Hence, if the data to be transferred is a change to a file which exists on system B, difference information Δ comprises only the differences in such file and where such differences occur. If the data does not exist at all on System B, the difference information Δ will be the entire file. Difference information Δ received by differencing receiver 102 at System B is reconstructed at System B, and the changes reflected therein are updated on System B. For example, if System A and System B are two computers and an update for certain binary files on System A is required, the differencing transmitter on System A will extract the differences in the file known to exist on System B and any new files, and transmit only those differences (an instructions for where to insert those differences) to the differencing receiver 102. Differencing receiver 102 will interpret the difference information (Δ) and reconstruct the binary files on System B. In this manner, the information on System B is updated without the need to transfer the entire binary files between the Systems.

Fig. 2 is a generalized block diagram of a second data transfer and synchronization system constructed in accordance with an exemplary embodiment of the present invention. In Figure 2, System A and System B include functional blocks 104, each representing a differencing synchronizer. The function of the synchronizer 104 is similar to that of the transmitter and receiver combined; the synchronizer will allow difference information Δ to be both transmitted and received. In one example, System A and System B are a portable computer and a desktop computer, respectively. When information such as contact information is to be synchronized between the two, the differencing synchronizer 104 will extract changes made to the contact information on either System A or System B and at predetermined times, transmit the information Δ between the systems, and reconstruct the data on the receiving system to update information from the sending

system, in order to ensure that both systems contain the same data.

Fig. 3 is a generalized block diagram of a third data transfer and synchronization system constructed in accordance with an exemplary embodiment of the present invention. System A again includes a differencing transmitter and System B includes a differencing receiver 102. In this embodiment, a storage server 300 is coupled between System A and System B. Storage server 300 may store a separate database of the difference information Δ provided by System A, which allows System A to provide its difference information Δ to the storage server 300 at a first point in time, and storage server 300 to provide the same difference information Δ to System B at a second point in time, but not the same as the first point in time. In addition, multiple sets of difference information Δ may be provided at different points in time, and stored for later retrieval by System B. Still further, the difference information sets may be maintained on server 300 to allow data on either System A or System B to be returned to a previous state.

Once again, the storage server 300 is coupled by a direct connection 110 to both System A and System B. Storage server 300 may be a server specifically adapted to receive differencing information Δ from the receiver 100 and provide it to the transmitter 102. In one embodiment, server 300 includes specific functional routines for enabling this transfer. Alternatively, server 300 comprises standard information server types which respond to standard Internet communication protocols such as file transfer protocol (FTP), or hypertext transfer protocol (HTTP).

Fig. 4 shows yet another alternative embodiment of the system of the present invention wherein System A and System B, once again coupled directly to a storage server 300 by a direct connection line 110, each include a differencing synchronizer 104. Difference information Δ can be passed to and from System A through synchronizer 104 to and from the storage server 300 at a first point in time, and to and from System B at a second point in time. In this embodiment, storage server 300 includes routines, described below, for resolving conflicts between data which has changed on both System A and System B independently after the last point in times when the systems were synchronized.

Fig. 5 shows yet another exemplary embodiment of the present invention including four systems: System A which includes a differencing synchronizer 104; System B which includes a differencing receiver 102; System C which also includes a differencing synchronizer 104; and System D which includes a differencing transmitter 100. Each is directly coupled to a storage server 300, allowing control of transmission of differencing data Δ between the various systems. Server 300 may include routines, described in further detail below, to track the various types of systems which comprise System A through System D, and which control the transmission of various components of the difference information Δ to each of the various systems. For example, since System B includes only differencing receiver 102, the difference information Δ_2 which is provided to it may be a sub-component of that which is transferred between System A in the storage server 300, or may be simply receiving broadcast information Δ_4 from System D. In one embodiment of the system of the present invention, server 300 does not itself route the difference information derived from each receiver/transmitter/synchronizer. Server 300 acts as a repository for the information, and the determination of which difference information Δ is attributed to which receiver/transmitter/ synchronizer is made by each receiver/transmitter/synchronizer.

Fig. 6 shows yet another exemplary embodiment of the present invention, in which a synchronizer is provided in storage server 300. It should be recognized that a forwarder and/or receiver may be provided in server 300 as well. The particular embodiment shown herein may be advantageous where device processing power and memory are limited, such as cases where the device is a cell phone. It should be noted that the data transferred between system A and the device engine 104a in such an embodiment may or may not be difference information, depending on whether System A has the capacity to detect and output difference information. Each of the devices may include a differencing receiver, a differencing transmitter, or a differencing synchronizer. It should be understood that a portion of the differencing synchronizer 104a may reside on System A and another portion may reside on server 300.

Fig. 7 shows yet another alternative embodiment of the present invention wherein a plurality of devices, such as those shown in Fig. 6 are coupled to a combination of public or private networks 700 such as the Internet. The network 700 includes one or more storage servers 300_1, 300_2, and in such cases the difference information Δ transmitted between each device via intermediate storage on one of such servers. Network 700 may couple the devices to one or more specialized function servers, such as servers specifically designed to include a differencing forwarder, receiver or synchronizer. The devices in Fig. 7 comprise, by way of example and without limitation, an office personal computer ("PC") 702, a smart telephone or cellular phone 704, a personal information Palm® computing device 708, a home PC 710, and a web browser 712. Each differencing receiver, differencing transmitter, and/or differencing synchronizer present in devices 702-712 includes means to poll the data stored on storage servers 300_1,300_2 to determine whether the data present at storage server 300_1,300_2 includes difference information which the particular receiver or synchronizer will use to synchronize the data on the device on which it resides.

In the following description, an embodiment wherein the differencing receiver, transmitter, and synchronizer are described will be discussed with respect to its use in synchronizing contact information, calendar information, and binary file information between a plurality of different devices in the context of data synchronization. It will be readily understood that the system of the present invention is not limited to synchronization applications, or applications dependent upon specific types of data, such as contact information or scheduling information. In particular, it will be readily understood that the transmission of data comprising only the differences in data between two systems via routines which extract the data and reassemble data on the various systems, represents a significant advancement in the efficient transmission of data. The present invention allows for optimization in terms of a reduction in the bandwidth utilized to transmit data between two systems, since only changes to data are transferred. This consequently increases the speed at which such transactions can take place since the data which needs to be transmitted is

substantially smaller than it would be were entire files transferred between the systems.

Generally, the system comprises client software which provides the functions of the differencing transmitter 100, differencing receiver 102, and differencing synchronizer 104 in the form of a device engine. The device engine includes at least one component particular to the type of device on which the device engine runs, which enables extraction of information from the device and conversion of the information to difference information, and transmission of the difference information to the storage server. This allows the replication of information across all systems coupled to the system of the present invention. Although the storage servers 300 utilized in the system of the present invention may be any type of storage server, such as an Internet server or an FTP server, and may be provided from any source, such as any Internet service provider (ISP), particular aspects of a storage server which may be useful and which may be customized to optimize transfer of information between systems coupled as part of the present invention will be described below. Synchronization of devices utilizing the synchronization system of the present invention is possible as long as an Internet connection between the devices is available. The Internet connection between the devices or between the devices and a server, need not exist at the same point in time, and new devices may be added to the system of the present invention at any point in time without the loss of information. The system provides totally transparent access to information and the device engine on each device provides an operating system independent extension which allows seamless integration of the personal information services in accordance with the present invention. In addition, only those changes to the information which are required to be forwarded to other systems on the system of the present invention are transmitted to enable exceptionally fast response times. In a still further aspect of the invention, information which is transferred in this manner is encrypted to ensure security over the public portions of the Internet.

Fig. 8 is a generalized block diagram of the system architecture of a data transfer and synchronization system constructed in accordance with an exemplary

embodiment of the present invention. In this embodiment, the system of the present invention allows the coupling of a collection of personal devices and applications one uses when working with personal information. Nevertheless, the system may be used to broadcast public or private information to various device types. System software in the form of a device engine for each device which is declared a part of the system of the invention is distributed across the collection of devices to enable synchronization. Distribution of the device engines may occur via, for example, an installation package forwarded over an Internet connection. In essence, the device engine software of the present invention forms a distributed processing network which maintains consummate synchronization of all information in the system. The processing load associated with delivering this service is pushed to the end-point devices which provides for easy scaling of the system to ever-larger applications.

In Fig. 8, two types of device engines are shown. One type is situated on the various devices and outputs change data to the server; and the other type is embodied on the server and receives device-generated change information from the device. An alternative exemplary embodiment includes a hybrid of the two, that is, a portion of the device engine is on the device and a portion on the server.

As shown in Fig. 8, any number and type of devices 802-808 may be utilized in accordance with the system of the present invention. A telephone 802 may comprise a cellular phone or a standard POTS-connected telephone. Telephone 802 may include contact information and, as is supported with a newer generation of cellular telephones, appointments and task data stored in a data structure 812. The application 812 which utilizes the application data 822 comprising such information is all stored in the telephone unit 802. Likewise, a personal digital assistant such as a Palm® computing device 804 includes application 814 and application data 824 which may include information such as contacts, appointments and tasks, and may also include file information such as documents which are created and stored on the PDA 804. Device 806 is represented as a Windows personal computer running an operating system such as Microsoft Windows 95, 98, NT or 2000. Applications 816 which may be running on device 806 include the Windows operating system itself,

Microsoft Outlook, Symantec's ACT Personal Information Manager, Goldmine Software's Goldmine, Lotus Organizer, Microsoft's Internet Explorer web browser, Netscape's Communicator Suite, Qualcomm's Eudora e-mail, and various other programs, each of which has its own set of application data 826 which is required to be synchronized not only with devices outside the system 806, but also between devices and applications within the system itself. Finally, a dedicated web browser client 808 is shown which couples via the Internet to web portal applications 816 which have their own set of application data 828. Unlike devices 806 which store the application and application data substantially in their own hardware, web portal applications are provided on a separate server and provided to browser 808 via an Internet connection. Nevertheless, the web portal application stored on the portal application provider includes a set of application data 828 which a user may wish to synchronize. For example, a large web portal such as Yahoo! and Snap.com provide services such as free e-mail and contact storage to their users. A user may wish to synchronize this with applications running on their cellular phone, PDA, or Windows devices.

In order to access the specific application data of each of the systems shown in Figure 8, a device engine is associated with each type of device. A cellular device engine 862 communicates and incorporates itself with the application data 822 of the cellular phone. Likewise, a PDA device engine 864 is provided, which may be based on either the Palm® operating system, Windows CE operating system, or other PDA-type operating systems as necessary. A Windows-based device engine 866 includes a mechanism, discussed below, for extracting application data 826 from supported Windows applications 816, and a web services device engine 868 incorporates to extract application data 828 from web portal applications 818.

As shown in Figure 8, some device engines are provided entirely on the device (and are referred to herein as desktop device engines), while others include components a the back end server (which may comprise storage server 850 or a specialized server, as shown in Figure 9B.) This is illustrated generally by lines 832, 834,836, and 838 in Figure 8. Also, in Figure 8, elements above dashed line 855

are provided by an administrator or service provider of the system of the present invention. Each of the device engines 862, 864, 866 and 868 is configured relative to the type of device on which it resides. For example, the Cell phone device engine 862 includes one or more components arranged on the phone while others are on server 850. Conversely, device engine 866 resides entirely on the windows device 806.

Data from each of the devices is coupled via an Internet connection 710 with a storage server 850. As noted above, storage server 850 may be a generic storage server or it may be a storage server specifically adapted for use with the system of the present invention as discussed below. One or more of the storage servers 850 are used to communicate transactions amongst the collection of systems 802, 804, 806, 808. It should be readily recognized that any number of different types of systems 802, 804, 806, 808 may be provided in accordance with the present invention and incorporated into the system. However, for brevity, not all the different types of commercially available computing devices which are currently in use or in development, in which the system of the present invention may be incorporated, are listed.

In its simplest embodiment, the storage server 850 is simply a dumb storage server and each of the device engines transmits only difference information thereto to be stored in a particular location accessible by other device engines in the system. In one embodiment, each device engine implements all processing required to keep all the systems fully synchronized. Only one device engine needs to be coupled to the storage server 850 at one particular point in time. This permits synchronization of multiple systems in a disconnected fashion. Each device engine will download all transactions encapsulating changes that have occurred since the last synchronization from the server and apply them to the particular device.

The change or difference information (Δ) is provided in one or more data packages, the structure of which is described herein. Each data package describes changes to any and all transfer information across all device engines, including but not limited to application data, files, folders, application settings, and the like. Each

device engine can control the download of data packages that include classes of information that apply to the specified local device 802, 804, 806 or 808 attached to that specific device engine. For example, device engine 862 will only need to work with changes to information describing contact names and phone numbers in application data 822, while device engine 866 will be required to work with changes to e-mail, changes to document files, notes, as well as contact and address information since the application data 826 is much more extensive than application data 822.

Each device engine includes compression/decompression and encryption/decryption components which allow encryption and/or compression of the data packages transmitted across Internet connection 710. It should be recognized that compression and encryption of the data packages may be optionally provided. It is not required in accordance with the present invention. Each device engine performs mapping and translation steps necessary for applying the data packages to the local format required for that type of information in the application data stores 822-828. The device engine also includes components which allow it to track ambiguous updates in cases where users have changed data to a particular data field on two different systems simultaneously since the last update. In this case, the device engine includes a mechanism for drawing this to the attention of the user and allowing the user to resolve the conflict.

Fig. 9A illustrates an exemplary device engine utilized with a generic application 810 and a generic storage server 850. In particular, the device engine of Fig. 9A is a desktop device engine, since all processing occurs on the device and only difference information is transmitted to server 850. Nevertheless, an understanding of the desktop device engine will aid in understanding server side devices engines, hereinafter described. Shown in Fig. 9 are the functional components of a device engine in block form and their interrelationship to each other. The device engine 860 is equivalent to the functional block of a differencing sequencer 104 shown in Figures 1-7. Portions of the functionality are used as needed in a forward-only (a differencing transmitter) or a receive-only (a differencing

receiver) capacity, as required by the particular application.

A device engine exists for each and every device that makes up a user's personal information network of devices in the system. As shown in Figure 9A, each device engine 860 includes an application object 910. The application object is specific to each particular application 810 and provides a standard interface between the device engine and the balance of the data transmission system of the invention, and the application 810. Details of the application object will be described in further detail below. The application object is a pluggable architecture which supports a wide variety of vendor-unique applications. The job of the application object is to map data from the application into a temporary or "universal" data structure by connecting to the application via any number of standard interfaces to gain access to the applications data. The data structure of the application object puts the data in a generic or "universal data" format which may be used by the device engine components to generate data packages for provision to the storage server.

Also provided is an application object store (AOS) 920 which includes a copy of the device's data at a point just after the previous data extraction and synchronization occurred. Application object store 920 is a mirrored interface which stores a snapshot of the previous state of the data from the application object 910 in the device engine. The size of the AOS will depend on the data being collected by each device engine.

The generic output of the application object is provided to a delta module 950. Delta module 950 is a differencing engine which calculates differences in data between the output of the application object 910 and the copy of the data which is provided in an application object store (AOS) 920. The actual differencing and patch routine can comprise a routine such as XDelta or YDelta. The delta module 950 will be referred to herein alternatively in certain portions of the description as "CStructuredDelta." In addition, the difference information is alternatively referred to herein as a "change log." Each change log (or set of difference information) is a self describing series of sync transactions. As described below, the change log may be encrypted and compressed before output to the network.

Hence, during a sync, the Application Object will, using a mechanism discussed below, extract the data of each application in the device and convert it to a universal data format. The delta module will then generate a difference set by comparing the output of the Application Object and the AOS. This difference information is forwarded to the encryption and compression routines for output to the storage server 850 in the form of a data package. Alternatively, the data from one application can be used to synchronize to data in another application in, for example, a windows environment, as shown by arrow 1050 in Figure 10.

It should be specifically noted that the application object may interface directly unstructured binary data or with structured application data. The differencing routine supports both uses of the delta module 950 in comparison generation.

In some cases, operation of the application object and delta module is simplified by the fact that some applications, such as PDA's, have the ability to output changes to its data. In such cases, the delta module 950 need only provide the data into the data package, since comparison to an AOS is not required – the application already includes a mechanism for tracking changes made to its own data. However, in many cases the applications provide, at most, a standard interface to access the data, such as Microsoft's OBDC interface, the Microsoft standard Application Programming Interface (API), or other similar standard interfaces.

Device engine 860 further includes a versioning module which applies a version number per object in the data package. As explained further below, each object in the data package is assigned a universally unique ID (UUID). Hence, unlike many prior synchronization systems, the system of the present invention does not sync data solely by comparing time stamps of two sets of data. Versioning module 915 allows each device engine to check the state of the last synchronization against data packs which have been provided to the storage server to determine which data packages to apply. This allows the device engine to sync itself independently of the number of times another device engine uploads changes to the storage server. In other words, a first device engine does not care how many times

a second device engine uploads data packages to the server.

An events module 925 controls synchronization initialization events. Items such as when to sync, how to sync, trigger the delta module 950 to perform a synchronization operation.

A user interface 930 is provided to allow additional functional features to a system user of the particular device to which the device engine 860 is coupled. The user interface is coupled to a conflict resolution module 940, a filtering module 945, and a field mapping module 935. Each of the modules provides the functionality both necessary for all synchronization programs, and which users have come to expect.

Filtering module 945 allows filtering for types of content based on, for example, a field level content search. The field mapping module 935 allows for the user to re-map certain interpretations of items which were provided in the document stream. For example, if the device engine 860 is operating on a personal computer, and a synchronization is occurring between the personal computer and a notebook computer, and the user has a "my documents" directory on the personal computer which he wishes to map to a different directory on the notebook computer, the field mapping module 935 allows for this re-mapping to occur. It should be recognized that the field mapping module allows for changes in directing the output of the data package. The field mapping module 935 is not necessary to map particular data fields of, for example, contact information from one application, such as Microsoft Outlook, to a different application, such as Symantec's ACT, as is the traditional use of field mapping and synchronizing applications.

Delta module 950 is further coupled to a compression module 970 and an encryption module 960. It should be recognized that the compression encryption modules need not be enabled. Any type of compression module 970, such as the popular PK Zip or Winzip modules, or those available from HiFn Corporation may be utilized in accordance with the invention. Moreover, any type of encryption algorithms, such as MD5, RCH 6, Two Fish, or Blowfish, or any other symmetric encryption algorithm, may be utilized. In one embodiment of the invention,

encryption without compression is used. In a second embodiment of the invention, compression without encryption is used. In a third embodiment of the invention, neither compression or encryption is used, and in a fourth embodiment of the invention, both compression and encryption are used.

Versioning module 915 also allows the device engine 860 to support multiple users with distinct synchronization profiles. This allows multiple users accessing the same machine to each synchronize their own data set using the same device engine. For example, if the application 810 on a particular device comprises Microsoft Outlook on a personal computer, coupled to a Microsoft Exchange server, and Outlook is configured to have multiple user profiles, versioning module 915 will track the data applied through the device engine when a sync request occurs. This allows two users of the same Outlook client software which access different data sets, either in the client computer or on a separate server, to utilize the same device engine and the system of the present invention via the same machine. In a further embodiment, a particular device engine supports the use of foreign devices accessing the system via the same connection. Palm® devices, for example, use a cradle to connect to a computer and/or Internet connection. If a particular user wishes to allow another user to use his Palm® pilot cradle connection to synchronize the other user's Palm® pilot, the device engine can generate data packages to update the local application object store for the foreign device. The application object store can therefore be used as a temporary storage for cases allowing synchronization of foreign devices.

The output of the device engine 900 comprises a data package which is output to storage server 850. As noted above, only one device engine need be connected to the storage server 850 at a given time. The data package can be stored on the storage server 850 until a request is made to a particular location of the storage server by another device engine. Likewise, delta engine 900 can query alternative locations on the storage server for access to synchronized data within the system of the present invention. Access to areas of the storage server is controlled by a management server (MS) described more fully below. In one embodiment,

each sync operation requires that the device engine for each device login to the management server to authenticate the device and provide the device engine with the location of the individual device's data packages on the storage server.

Data packages may be advantageously provided to the device engine from the storage server in a streaming format, allowing processing to occur using a minimum of bandwidth and storage in the devices. The device engine 860 and particularly the delta module 950 interpret data packages based on the versioning information and the mirrored data present in the application object store 920. When data is returned to the delta module 950 from the storage server 850, the delta module returns differenced data to the application object 910 for the particular application which then translates the delta information into the particular interface utilized for application 810. Once a device engine has been fully applied all data packages from an input stream, it generates a series of data packages that describe the changes made on the local system. The device engine uses the local application object store 920 to keep track of the last synchronized version of each application's actual data, which is then used for the next data comparison by the delta module on the next sync request. Generated data packages can include operations and encode changes generated from resolving ambiguous cases as described above.

Figure 9B depicts how server based device engines may be provided in the system of the present invention. The Palm® device example is shown in this embodiment, where the Palm® device has the capability of connecting directly to the Internet and a service provider's data center 900. The data center includes a firewall 975 to prevent unauthorized communications with servers resident in the data center 900 and protect integrity of the data. The storage server 850 may communicate directly through the firewall as may the management server (MS) 1410. Shown therein are two sync servers 982 and 984 each of which is dedicated to syncing one particular type of application. Sync server 982 is dedicated to the Palm® device, while sync server 980 is dedicated to, for example, a portal application (Portal1).

Since the Palm® Device 804a includes a mechanism for transmitting changes to its data directly, data may be transmitted using HTTP request and response via

the firewall 975 to the sync server 982 where differencing and updating of data in the AOS can occur, after which changes can be downloaded to the Palm® 804a.

The synchronization server is an application handles concurrent synchronization of user's data. Each Sync Server includes plug-in support for multiple devices to be synchronized using the same sync server executable. Each device type has it's own device name that identifies which AO / AOS components will be used during the sync.

The sync server uses the concept of a universal data record in its internal sync differencing engine and when sending data to and retrieving from external entities such as the AOS and AO. Hence, in the Palm® application, the job of a server AO is simply to take the device-specific format of its record and convert into a universal record format.

The Sync Server has a plug-in architecture so that 3rd party application partners can easily add their services into the server. Currently, if the server is operated in a Microsoft Windows NT Server, the sync server discovers the sync components via the Windows NT registry. In alternative embodiments, this function is performed in a Component Manger which operates on each sync server to manage processing by each of the AO and AOS on the server. Each AO and AOS are implemented as a stand-alone DLL that the Sync Server loads at initialization time, or when adding a new component via the Component Manager.

Each sync server is shown as dedicated to a single application. However, a sync server may handle multiple device types.

In the embodiment of Figure 9B, it should be noted that, depending on the device type, there are different configurations for the AOS and AO's. For example, the Palm®'s AO data store 1050 resides on the Palm® device 804a itself and a separate AOS data store 1052 exists for this configuration (an Oracle database). In the case of Portal1, the AOS and AO use the data store 1054.

Device engines can generate additional data packages intended to resolve synchronization problems in other systems. For example, interfacing with the conflict resolution module 940, if the user makes a change to a particular data store

on an application object on his Palm® pilot, then makes an additional change to a personal information manager (PIM) application on his personal computer, the user can specify that the change made on the personal computer will "win" when the conflict is detected by the Δ engine and the versioning information between the two devices. This is essentially a definition that one particular set of data is correct and should replace the second set of data.

Fig. 10 shows an exemplary embodiment of a desktop device engine used in, for example, a Microsoft Windows-based operating system environment. A Windows operating system may have at least three specific applications which may require synchronization. In Fig. 10, the system includes Netscape Communicator application 1040 having data such as bookmarks 1021, contacts 1022, and e-mail 1023; a Microsoft Outlook application 1042 which includes contact information 1024, calendar information 1025, e-mail information 1026, note information 1027, and tasks information 1028; and Windows operating system 1044 information including Favorites data 1029, file system information 1030, and individual files 1031.

Each particular application 1040, 1042, 1044 has an associated application object 1010, 1012, 1014. Each of the respective application objects provides data back to delta module 950 in a generic format which is usable by the delta module in accordance with the foregoing description of the apparatus shown in Figure 9A. From Fig. 10, it will be additionally seen how the delta module 950 may be utilized to synchronize data between applications running on the same particular server. The device engine hence does an intra-system sync such as, for example, between the contact information 1022 from Netscape and the contact information 1024 from Outlook.

Fig. 10 further illustrates the modularity of the system of the present invention allowing the device engine to include any number of different application objects to be provided on a single device to incorporate all applications run on that device. In operation, during an installation of a device engine into a particular system, the installation program may be tailored to provide application objects which may be present on a given system. For example, the installation program for a Windows

machine will carry any number of application objects for systems and applications which may be present on a Windows machine. The installer will check for the presence of given applications, and allow the user to add additional applications which may be installed in locations that are not the normal default installation areas for application support by the application objects which the installer is carrying, or de-select certain applications which, for one reason or another, the user may not wish to install an application object for and render a part of the system of the present invention.

In order to provide security and identification of particular users in an Internet-implemented synchronization system, a management server may be provided in the system of the present invention. The management server is a centralized server which controls behavior and characteristics of the entire network of device engines across all users.

Fig. 14 shows a general representation of a management server 1410 integrated into an exemplary system of the present invention. Also shown in Fig. 14 is an exemplary device engine 1450 which has HTTP links to both management server 1410, a storage server 1415, and a generic FTP server 1420. As will be discussed hereinafter with reference to the process of the present invention, and the specific implementation of the data below shown in Figures 15-17, the management server interacts with the device engine to control authorized access to information on the storage server, or a generic FTP server 1420,1425 to access device-specific information storage 1430 in accordance with the system of the present invention. This allows any device coupling to the Internet to have access to management protocols and to retain user information across all platforms which the data which is being synched by the system of the present invention must access. The management server preferably communicates using hypertext transfer protocol (HTTP) which may be implemented with a secure sockets layer (SSL) to ensure security. The management server supports an authentication interface that requires each device engine to authenticate with the management server before performing synchronization. Certain storage server implementations may utilize

locking semantics to control read and write access to storage for multiple device engines. For example, in a generic FTP request, if two device engines attempt to connect to the same data at the same time, there must be some form of locking control to prevent device engines accessing the same data at the same time. In this instance, the management server controls the device engine acquisition, renewal, and releasing of locks against data stored in the network.

Each device engine is uniquely identified and tracked by the management server. This allows for tailoring behavior between the management server and specific types of storage systems and device engine components. All device engine components are tagged and version stamped for management via the management server.

Device actions can request updated copies of individual device engine components, permitting self-update and configuration of device engine systems. This permits minimal download designs for device engines that are on low bandwidth connections enabling the device engines to download additional required components at a later time.

In a further aspect of the system, a value added component may be provided where the management server can support client's advertising mechanisms, enabling the display of banner or similar advertising on a device engine system without the need for a web browser. Cycling of advertisements, statistic collection, and the like, are managed via management server protocols. Online purchase and subscription mechanisms are also supported using the management server protocol.

The management server further supports the accounting, sign-up registration, device edition, storage server selection, and similar functions for each user in the system. In one embodiment, the management server may retain password and encryption information for a given user account. In a second embodiment, such information is not retained. The second embodiment provides the advantage that users may feel more secure if the maintainer of the management server is not in possession of the password to access data in the user's account.

Further information with respect to the management server and the data flow

from the management server to other components of the system of the present invention will become apparent with respect to the discussion of the process flow and data flow diagrams in Figures 15-17.

Figure 17 shows a general depiction of the data flow and the functional specification of the management server utilized in accordance with the present invention.

As shown in Figure 17, following a welcome request 1710, a user is allowed to sign out which enables an add user module 1712, and subsequently enables an add device module 1714. If sign-up is not requested, information may be provided via module 1718.

As indicated in Figure 17, the add user module 1712 adds user records to the user in device database 1750. Additionally, the add device module 1714 adds users and devices to the user device database 1750. A device list 1720, and a device engine download and update database 1722, provide selection data for the add device module 1714. The account authentication module 1724 receives input both directly from a user log-in from the welcome screen at 1710 and from the add device module 1714.

Once an account is authenticated and confirmed, the administrator of the system of the present invention having a private data store at 1770 may choose to provide a web desktop 1754 which allows access to a user's records such as file 1756, e-mail 1758, calendar 1760, contacts 1762, notes 1764, and tasks 1766. The information will be culled from a provider database 1752 which will be synched in accordance with the system of the present invention as previously described. In essence, the provider database 1752 accesses data from the device engines 1780, which include, as discussed above, the storage server, each individual device engine 1785, and a settings database 1787.

Other portions of the management server include the locking modules for beginning a sync 1732, continuing a sync 1734, and ending a sync 1736, and for updating user information including modifying a user 1742, adding devices 1744, removing devices 1746, and modifying devices 1748.

Shown in Fig. 14 is an exemplary storage server 1415. While storage server 1415 may include a generic storage model accessible through any number of standard Internet protocols, in accordance with the present invention, a flexible storage architecture is provided that permits various standard implementations of the system of the present invention. This allows deployment of network services without installation of new server applications and can be responsible for communicating change information between multiple device engines in a consistent fashion.

One or more storage servers 1415 may be used to communicate transaction amongst a collection of devices. Each user's personal information network is represented by a unique account within its own data package storage section. The storage server 1415 maintains persistent store collection of data packages which is, at a minimum, enough data packages to be capable of synchronizing the most out-of-date system in a user's given information network or add information to new devices which are provided in the network. Additional data packages can be maintained to permit rollback of previous versions of information. The storage server can automatically dispose of older data package storage and can support aging of an inactive accounts.

Each storage server 1415 may be implemented using a variety of implementations including a standard FTP server for any operating system platform. The storage server can be implemented using HTTP protocols for increased efficiency and firewall avoidance. The storage server may be implemented using techniques for local storage such as database access or single file storage of a user's entire file system tree. The storage server 1415 may utilize the stored foreign protocol model for moving copies of data packages to other storage servers in the system. In one embodiment, the storage server can allow tunneling of information using an alternative protocol to other storage servers in cases where firewall prevents originating protocol. For example, a storage server can relay an FTP traffic inside an HTTP protocol. Storage servers may include their own locking semantics to arbitrate multiple device engine access to the same server without the need for a separate management server. Each device engine can access only a specific user's

data package storage area even though the storage server 1415 may maintain a larger number of data packages across a large number of users. This allows for increased scaling when the storage server is implemented using file system techniques.

In one aspect, the storage server is implemented using standard FTP or HTTP connections for each operation. HTTP is composed of request response pairs. All requests are supposed to be posting commands. Parameters can be set in the form known as "application/X-WWW-form-URLENCODED". The encoding is specified as in RFC1866. Functions for the storage server include testing if the storage server can reach other users which will retrieve a simple text string, a "get" command which transfers the contents of a file as in a binary stream of byes; a put command as a binary stream of data to the storage server, a directory listing command, a remove command, a rename command, an exist command, and the like.

Figure 15 represents a "pull" synchronization process in accordance with the present invention. Both the pull synchronization illustrated in Fig. 15 and the push synchronization illustrated in Fig. 16 are done from the perspective of the device engine. A pull synchronization as illustrated in Figure 15 is preferably performed prior to a push synchronization. This allows the device engine to know whether synchronization of its own data is necessary.

Each device has its own triggering mechanism for initiating synchronization. Some devices, such as Windows clients and Palm® pilots are triggered manually when the user presses a "sync" button. Other devices, such as a cellular telephone, may be triggered automatically after another device completes a sync. Regular, time-based triggers are supported as well. A web-based application portal will sync when a user logs into the website security authorization mechanism, and may optionally sync on a log-out of the user or on the session time-out, but only if the user has changed data during the session. For each sync, the triggering event specifies which application types are to sync for the device. This enables a triggering event to trigger only a sync for a particular application type. The

management server can specify that no sync is needed for a particular type of application to minimize traffic to the storage server. Syncs may be triggered via an HTTP request to the server. This request holds information about which device to sync and the user log-in information is bounced to the management server for authorization and validation. Syncs may be triggered by sending an HTTP request to the server and passing the authentication information in the data portion of the request to the management server. Each device may include a servlet that is responsible for retrieving the request and ensuring its proper format before passing the synchronization request on to the server.

The device name and device class uniquely identify a particular device type that is being synchronized, and is contained in the management server. Each user has one or more device entries in the management server authorization records and each device name is unique for this user's space. For example, if a user has five devices with his or her own personal identification number, there will be five authorization records. There may be two Windows devices, two different Palm® devices and a web service portal, each having their own personal identification number.

As shown in Figure 15, the pull synchronization process starts at an idle state 1405 when the triggering event, described above, triggers a synchronization request. The synchronization request is confirmed at 1410 and if the request is verified, a connection is made to the storage server at step 1415. Once a connection is established, the connection to the management server is made at step 1420 to authenticate the user identification via the management server. If authentication is successful, the management server may initiate a management server lock on the storage server so that no conflicting device engines may couple to the same data at the same time. A failure at any of the steps 1410-1425 will return the system to its idle state 1405. Once the engine server lock is acquired, the storage server will be checked to determine whether a new version of the data exists on the storage server at step 1430. If no new version exists, the synchronization process ends.

If a new version of the data exists, the device engine will retrieve the

difference information at step 1435 "o get Δ".

Once a Δ is retrieved, conflicts are resolved at step 1450. The resolve conflicts step allows a user to resolve conflicts to multiple types of data which have been changed on both the server portion of the device and in the local data.

Once the conflicts have been resolved at step 1450, the Δ's are applied at step 1455. The apply Δ step 1455 allows for filters and mappings to be accounted for on the local device engine side of the system. As shown at steps 1460, 1465, 1470, and 1475, the Δ may include updates at the item level 1460, application level 1465, device level 1470, or network level 1475. In each of the aforementioned steps, a loop back to the Δ retrieval step 1435 is provided. When no further Δ's are available, the management server lock is released at step 1440.

Fig. 16 shows an exemplary push synchronization in accordance with the system and method of the present invention. Beginning at idle state 1505, a synchronization event occurs and if confirmed at step 1510, Δ's are checked at step 1515. Depending on which type of changes occurred, a network Δ 1520, device Δ 1525, location Δ 1530, or item Δ 1535 will be created.

Once the Δ's for a given application have been created, the method of the present invention continues at step 1540, which enables a connection to a storage server. Upon connection to the storage server, a further connection to management server 1545 will occur to authenticate the user in the system. Failure at any of the aforementioned points will result in returning to idle state 1505. Upon authentication, a management server lock is enabled to ensure that multiple device engines do not connect to the same data at the same time.

Once a lock is acquired at step 1555, Δ's are uploaded to the system. As shown, this may include uploading an item Δ 1575, an application Δ 1570, uploading a device Δ 1565, or a network Δ 1560. Once Δ's have been uploaded to the server, management lock server 1580 is released, and the connection to the storage server is terminated at step 1585.

It should be recognized that such a push synchronization need not occur directly to a server, but may occur directly to a second device engine in accordance

with the depiction of the multiple embodiments of the invention in Figures 1-7.

Once information is provided into the universal data format, the device engine organizes the format into a data package. Each data package thus includes a description of changes to any and all information for particular application, and a collection of data packages describes changes across all device engines including all different types of data. With encoding and compression, data packages can become very compact to minimize bandwidth and storage requirements across the system of the present invention.

In one particular aspect of the present invention, encoding of the data packages may be provided in a streaming format to allow processing by the device engines with minimal storage and memory configuration at the device engine level.

The device engine can read the stream and determine which records from which applications it needs to update the particular information present on the system on which it resides.

Data packages can be provided in a binary data format. This allows data packages to encode changes to non-application data at a bite level. Hence, if a single bit on a system changes, the system of the present invention allows synchronization of that bit on another system. Changes are described as a sequence of bite-level change operations. One such encoding is using a sequence of insert and copy operations. Insert and copy operations generally define a particular "insertion" of a number of bites from a source file, then how many bites of a changed source file must be inserted to a particular file, then how many bites to insert from a particular new file, with a differencing engine taking the bites in the stream and inserting them into the new file to create the new version of the file.

As will be readily understood by one of average skill in the art, this allows a user to, for example, change a binary file such as a word processing document or other type of attachment, and synchronize such an attachment at the binary level. Specifically, if one forwards an e-mail of a word document to a second individual, the second individual modifies it and wishes to return this document with modifications to the first individual, because the first individual has the original file on his system, if

both systems are enabled in the system of the present invention, the second system need only send the changes or the difference information back to the first system in order for the first system to reconstruct the document on the second system using this change data to create the document as intended by the second user.

Multiple caching of both the generation and application of data packages can be utilized to deal with communication issues in accordance with the system of the present invention. It should be further recognized that data packages can be merged into larger meta-data packages. Such meta-data information, such as the organization of multiple device packages, may be encoded into a larger system package. Each system package is essentially an encoded sequence of data packages.

Figure 12 shows the general format of the data package and universal data format an object stream hierarchy used in accordance with the present invention. With reference to Figures 11 and 12, one will note that each item in a particular application data structure will have a particular classification, such as a file, folder, contact, e-mail, calendar, etc. as shown in Figure 13. The universal data structure contains a mapped item field for each type of data possible from each application supported by the system. Hence a "master" list of every data field mapping possible will contain a large number of items. Each application object requires a subset of such fields. One exception is an application object used for a Web portal application which provides access to all information available on all devices, including other Web portals.

Particular examples of item fields 1260 which may be included for any given item 1250 are shown in Figure 13. These exemplary item objects may, for example, be from an allocation such as Microsoft Outlook. Outlook allows for note items 1310, e-mail items 1320, task items 1330, calendar items 1340, bookmark items 1350, file items 1360, channel items 1370, folder items 1380, and contact items 1390, all of which have fields such as those represented in Figure 13.

The data format also contains folder information 1240 which allows the classification of items and consequently their associated item fields into particular

categories.

Application objects 1230 include information on the types of applications from which information in the stream is included. Device objects 1220 include information on the origin type of device which the information is originating from. Network objects 1210 include information on a user level to define that the information in the data stream is coming from a particular user.

As detailed above, each application object supports a folder store interface that permits management of collections of information on a folder level, and permits management of folder hierarchies of information. The application object also includes an item interface that permits management of individual information entries such as records or files or components of information entries such as fields within records. Each application object further supports an interface for detection of a vendor application.

A DataPack essentially contains a sequence of transactions describing changes to information. This information can span two basic types: structured or application data, and unstructured or binary file data. Transactions are encoded using an efficient streaming format with tags to represent the actual content objects. This technique permits the continuous extension of the DataPack format as new content is supported.

The general architecture of the package provides for transactions, application data, file data, files, objects and identifiers to be carried in the data package. Generally, transactions, application data, file data, and files have previously been described.

The first portion of the data package will be the data package identifier. Each transaction has a basic architecture of objects and operations. Each piece of content is referred to as an object and is uniquely represented with a Universally Unique Identifier (UUID). Objects typically are represented by a dynamically generated UUID, but more common objects are represented by static UUIDs. Each UUID preferably has a unique 128 bit value which may be assigned by the system provider.

Transactions are broken down into manageable blocks in the form of individual files. These files are then optionally compressed and encrypted and prefixed with appropriate headers. Transactions are grouped into specific files based on the following rules:

- Transactions related to account information are grouped into a DataPack file.
- Transactions related to a specific data class are grouped into a DataPack file.
- Transactions referring to binary data are grouped into separate DataPack files for each file object.

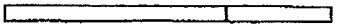
A DataPack file is identified using specific rules based on the file name. The file name is of the form "UUID.VER" where UUID is the identifier for the specific object and VER is the transaction version number. The version number is of the form "D0001" with additional digits used for large version numbers. The "D000" value is preferably reserved for the base version for the object.

The UUID for the user account is generated by the Management Server (MS). The MS also maintains a current table of UUID values and version numbers that provides the root structure for understanding the DataPack files within a user account. The MS also provides necessary locking semantics needed to maintain consistency when multiple device engines attempt to synchronize.

All DataPacks are prefixed with a standardized header that provides basic content information regarding the DataPack. Compression and encryption headers follow the DataPack header if needed.

The data package header information will include version signature, applied versioning information, content type, Δ engine type, compression type, encryption type, applied size, encrypted size, compressed size, raw data size, and other data useful for the device engine in decrypting the data stream to provide the data into a format usable for the application.

The header may optimally have the format:



Type	Bytes
Version	4
Signature	4
AppliedVersion	8
ContentType	4
DeltaType	4
CompressionType	4
EncryptionType	4
AppliedSize	4
EncryptedSize	4
CompressedSize	4
RawSize	4
Reserved	TBD

The following ContentType values are permissible:

Field	Comment
DP_CONTENT_RAW	Raw
DP_CONTENT_COMPRESSED	Compressed
DP_CONTENT_ENCRYPT	Encrypted

ED	

The DeltaType encodes the type of binary file differencing used. The following DeltaType values are permissible using DataPackageDeltaType:

Field	Comment
PackageDeltaTypeUninitialized	Uninitialized
PackageDeltaTypeRawData	Raw binary data
PackageDeltaTypeDeltaXDelta	Xdelta binary difference
PackageDeltaTypeDeltaBDiff	Bdiff binary difference

The compression type specifies whether the DataPack has been compressed. A DataPack compression header follows the DataPack header if a compression type is specified. The following CompressionType values are permissible using DataPackageCompressionType:

Field	Comment
PackageCompressionTypeUninitialized	Uninitialized
PackageCompressionTypeNone	None
PackageCompressionTypePK	PKZip format
PackageCompressionTypeLZS	LZS format

The encryption type specifies whether the DataPack has been encrypted. A DataPack encryption header follows the DataPack header if an encryption type is specified. The following EncryptionType values are permissible using DataPackageEncryptionType:

Field	Comment
PackageEncryptionTypeUninitialized	Uninitialized
PackageEncryptionTypeNone	None
PackageEncryptionTypeXORTest	XOR masked data
PackageEncryptionTypeBlowFish	Blowfish
PackageEncryptionTypeTwoFish	Twofish

All DataPack compression headers are encoded using the following format:

Field	Size (bytes)	Comment
Size	4	Size of data including this header
Version	4	Version (1)
Signature	4	Signature (4271)
HeaderType	4	Header type (HeaderTypeCompression)
Reserved	12	Reserved
DecompressedSize	4	Decompressed size

Reserved	50	Reserved
Reserved	12	Reserved

The following HeaderType values are permissible using DataPackageHeaderType:

Field	Comment
HeaderTypeUninitialized	Uninitialized
HeaderTypeEncryption	Encryption header
HeaderTypeCompression	Compression header
HeaderTypeRaw	Raw header

All DataPack encryption headers are encoded using the following format:

Field	Size (bytes)	Comment
Size	4	Size of data including this header
Version	4	Version (6)
Signature	4	Signature (4270)
HeaderType	4	Header type (HeaderTypeEncryption)
Reserved	12	Reserved

DecryptedSize	4	Decrypted size
InitValue	16	TBD
KeyLength	4	TBD
ClearTextKeyBits	4	TBD
Salt	4	TBD
PadBytes	4	TBD
HMAC	20	TBD
Reserved	12	Reserved

The following Operation values are permissible using the Operation class:

Field	Comment
clNop	None
clAdd	Add
clDelete	Delete
clChange	Change
clMove	Move
clRename	Rename
clForceChange	Force change without conflict

The following FieldDataType values are permissible using clDataType:

Field	Comment

clInvalidType	TBD
clString	Unicode String bytes with a 32-bit length prefix
clString8	Unicode String bytes with an 8-bit length prefix
clString16	Unicode String bytes with a 16-bit length prefix
clEmpty String	TBD
clBlob	32-bit length followed by a byte stream
clBlob8	8-bit length followed by a byte stream
clBlob16	16-bit length followed by a byte stream
clEmptyBlob	TBD
clByte	8-bit value
clShort	16-bit value
clDword	32-bit value
clQword	64-bit value
clDate	DATE type (double)
clDouble	8 byte real
clFloat	4 byte real
clUuid	16 byte uuid
clZero	Zero value

clOne	One value
clUnspecified	Unspecified value
clDefault	Default value
clCollection	Collection with 32-bit length
clCollection8	Collection with 8-bit length
clCollection16	Collection with 16-bit length
clEmptyCollection	Collection with no length

Data package objects are organized into a hierarchy as follows:

```
Account::= DeviceList + DataClassList
DeviceList::= {Device}
DataClassList::= {DataClass} + ProviderList
ProviderList::= {Provider} + DataStoreList
DataStoreList::= {Folder} + ItemList
ItemList::= {Item} + FieldList
FieldList::= {Field}
```

An account is the root structure, which identifies information about the user's account. It may have exemplary field tags (eFieldTag_[NAME]) such as Name, Password, UserName and Version. The FieldTag ItemType value is specified as ItemType_PIN using enumItemType.

A device is a system identified as part of an account. Examples include PCs, handhelds, Web sites, and so on. It may have tags (eFieldTag_[Name]) such as: "name" and "type" and item type values (eDevice_[Name]) such as Portal, Palm, Windows, CellPhone.

A data class is a grouping of similar information types. Many data classes may be represented for a particular account. The data class may contain field tags (eFieldTag_[Name]) such as: Name; ItemType; SubType; IsManaged; Provider; Filter and Version.

The following ItemType values are permissible using enumDataClass

(eDataClass_[Name]):

Tag	Description
UNKNOWN	Unknown
CONTACT	Contact/address book
EMAIL	Electronic mail
CALENDAR	Calendar
TASK	Task/to do
NOTE	Note/memo
JOURNAL	Journal
BROWSER	Web browser favorites, cookies, etc.
FILESET	Collection of files
PIN	Account information
DEVICE	Device information
FILEBODY	Contents of file

A Provider is the application that maintains specific information within a data class. There can be more than one provider for a particular data class. Field tags include: Name, AppObjID, Password, Username and Version. Examples of provider tags permissible for the provider (eProvider[Name]) include: Portal, Palm®, MicrosoftOutlook®, Lotus Organizer, Microsoft Internet Explorer, Microsoft Windows, and so on.

Data stores are the containers for storing information within a provider. There can be more than one data store for a particular provider. Folders represent structural organization of information within a data store. Data stores are not required to support folders. Tags (eFieldTag_[Name]) supported for each data store include: Name, ItemType, IsManaged and OriginalPath. Item types permissible for the data store include: unknown; Folder; MAPI; Database and Store_File.

Folders represent structural organization of information within a data store. Data stores are not required to support folders. A folder is represented by a UUID and may contain any of the following field tags (eFieldTag_[Name]): Name; ItemType; IsManaged; FileAttributes; CreationDate; ModificationDate; AccessDate; SpecialFolderType.

The eFieldTag_ItemType value is specified as eItemType_FOLDER using enumItemType.

Items are individual informational components consisting of the actual user

data. They may contain field tags such as: Name, ItemType, IsManaged, and Version.

File items typically have the following additional field tags (eFieldTag_[Name]):

```
FileAttributes
CreationDate
ModificationDate
AccessDate
FileSize
FileBody
DeltaSize
Hash
```

Item types may take the format (eItemType_[Name]) and may include: extended; folder; attachment; contact; distlist; email; calendar; task; call; note; post; journal; form; script; rule; favorites; subscription; common_favorites; desktop; common_desktop; startmenu; common_startmenu; channels; cookies; programs; common_programs; startup; common_startup; sendto; recent; internet_cache; history; mapped_drives; printers; docs; doctemplates; fonts; window_settings; app_data_folder; app_settings; fileset; pin; device; data_store; file; provider; and data_class; internal.

A field is based on one of a set of base type definitions. All field tag information is encoded using the following format:

Field	Size (bits)	Comment
FieldTag	16	Unique tag number
FieldType	6	Field base type
FieldSubType	10	Field sub-type

A number of Field types are possible, including: unknown; long; dword; date; string; binary; float; double; collection; uniqueid; qword; uuid; file; invalid. LONG is a

four byte value encoded in big-endian format. FieldType DWORD is a four byte value encoded in big-endian format. FieldType String is a sequence of Unicode characters followed by a single NULL byte. Interfaces are provided with an MBCS value. FieldType Binary is a sequence of bytes. FieldType UniqueID is a sequence of bytes as defined by the Universally Unique Identifier (UUID) standard. AO interfaces are provided with a Locally Unique Identifier (LUID) value FieldType QWORD is an eight byte value encoded in big-endian format. FieldType File is a UUID that references a separate DataPack containing the file body data. AO interfaces are provided with a sequence of Unicode characters followed by a single NULL byte that describes the full path name for the file.

Any number of filed sub types are possible. Each of the sub-types includes all of the possible data types from all of the supported user applications. As should be well understood, the possibilities in the number of sub-types is quite large, and dynamic as each new application supported by the system of the present invention is added. Examples of sub-types include:

SubField Description	Description
Base	No sub-type specified
EmailAddress	Email address
EmailAddressList	Email address list
SearchKey	Search key
CategoryList	Category list
StringList	String list
DistributionList	Distribution list
Gender	Gender (enumGender)
TimeZone	Time zone (enumTimeZone)
Boolean	Boolean (TBD)
NonZeroBool	Boolean with non-zero value (enumNonZeroBool)
Priority	Priority
Sensitivity	Sensitivity (enumSensitivity)
Importance	Importance (enumImportance)
SelectedMailingAddr	Selected mailing address (enumSelectedMailingAddr)
TaskStatus	Task status (enumTaskStatus)
FlagStatus	Flag status (enumFlagStatus)
RecurrenceType	Recurrence type (enumRecurrenceType)
DayOfWeek	Day of week (enumDayOfWeek)
DayOfMonth	Day of month (1 through 31)
InstanceOfMonth	Instance of month (enumInstanceOfMonth)
MonthOfYear	Month of year (enumMonthOfYear)
BusyStatus	Busy status (enumBusyStatus)
AttachmentType	Attachment type (enumAttachmentType)
MailBodyType	Mail body type (enumMailBodyType)
RGB	RGB color value
ManagedState	Managed state (enumManagedState)
FaoId	FAO ID for provider

```
SpecialFolderType       Special folder type (enumSpecialFolderType)
ResponseState       Response state (TBD)
ResponseStatus          Response status (TBD)
JournalStatus       Journal status
PageStyle               Page style
PageNumberMethod        Page number method
DelegationState         Delegation state
MeetingStatus       Meeting status
MeetingInvitation       Meeting invitation
CalendarType            Calendar type
DateOnly                Date only
TimeOnly                Time only
PhoneNumber             Phone number
URL                     URL
FilePath                File path
PopMessageID            POP message ID
MIMEType                MIME type
INVALID                 All values must be below this
```

Fig. 18 is a generalized block diagram of an exemplary data transfer and synchronization system, including various components described above. In particular, the system includes network 700, as shown in Fig. 7. The network 700 includes one or more storage mediums, such as storage servers 300_1, 300_2, of Fig. 7. A plurality of devices, such as those shown in Fig. 7, are capable of coupling to network 700 and exchanging synchronization information in an off-line fashion, using the techniques described above. These devices include home PC 710 and office PC 702, both of which can be synchronized with one another using techniques as described above. Device engines as described above are coupled between the various devices and network 700.

In Fig. 18, client software is installed on both home PC 710 and office PC 702 and is configured to operate in conjunction with an operating system such as Microsoft Windows. The client software, when executed, interacts with the various applications on the user's PC. The user interacts with the client software and configures the software such that the applications are prioritized. Data is then extracted from the various applications, organized in a format independent of the particular application and device from which the data originated, and incorporated into a data package. With exemplary embodiments of the present invention, various classes of data are manipulated in this fashion, including contacts, bookmarks, and calendar events.

In one example, the program Microsoft Outlook is installed on home PC 710

of Fig. 18. In this example, ten contacts are programmed in Outlook. The user instructs the client software to synchronize the contacts using Outlook. The client software accesses Outlook, extracts the ten contacts, and assembles the contacts into a DataPack CONT.D000, where the UUID "CONT" identifies contact information as the specific object, and "D000" signifies that this DataPack is version "0." The contacts are combined in DataPack CONT.D000 as a collection of ten transactions, each of which is assigned a unique ID # 1, 2, . . . 10. For instance, ID 2 represents a contact for John Smith. In this example, each transaction 1, 2, . . . 10 has an associated action, "Add." DataPack CONT.D000 is then uploaded to the network 700 and stored on storage server 300_2.

Later, office PC 702 connects to the network and identifies DataPack CONT.D000. In particular, such identification includes office PC 702 sending a signal to a management server 1802, in this example, informing management server 1802 that office PC 702 has not downloaded any DataPacks for the particular data class, contacts in this example. The management server 1802 responds by sending a signal to office PC 702 indicating that a data package of change information for contacts has been stored on the server 300_2, since the last time office PC 702 connected to network 700. Office PC 702, in response, sends a signal to management server 1802 requesting the data package. The most recent data package(s) stored on server 300_1, are identified, in this example, version 0 of the contact data, CONT.D000. This DataPack CONT.D000 is then downloaded to office PC 702. The change information in that data package, "Adds" of contacts in this case, is then applied to the pertinent application in office PC 702. In this example, the client software on the office PC is configured to synchronize contacts using a Lotus Notes application. Thus, the ten Adds from CONT.D000 are applied to the contacts in Lotus Notes, so that the contact information in home PC 710 and office PC 702 is synchronized. The office PC 702 then sends a signal back to management server 1802 indicating that office PC 702 has applied version #0 of the contacts data package(s). This information is preferably maintained in a registry 1804 by management server 1802 for each and every device that couples to the

network to download and upload change information.

Subsequently, the user of office PC 702 updates the contacts in Lotus Notes and adds one or more contacts. In this example, 20 contacts are added. Thus, the Lotus Notes application uploads a second data package to network 700, the data package including the 20 contacts and each having the associated action, "Add." This data package represents, of course, more recent change information than the information in CONT.D000. The data package uploaded by office PC 702 is identified by a unique filename, in this example, CONT.D001. In addition, office PC 702 sends a signal to management server 1802 confirming that CONT.D001 has been uploaded to network 700. The registry 1804 is updated to indicate that office PC 702 is current with, that is, has already applied the change information in, CONT.D001.

Home PC 710 subsequently connects to data network 700, and the client software on home PC 710 communicates with management server 1802 coupled to network 700. In particular, home PC sends a signal to management server 1802 identifying CONT.D000 as the most recent version of change information the last time home PC 719 coupled to data network 700. The management server 1802 queries the storage servers for any more recent data packages of changes to contact information. The management server 1802 identifies such data package(s), CONT.D001 in this example, and sends a signal to home PC 710 informing home PC that such data package(s) exist. The client software on home PC 710 then requests the new data package(s), and management server 1802 then downloads the data package(s) to home PC 710. The change information therein, in this case the 20 contacts from CONT.D001 to be added, is then applied to the contact information maintained by Microsoft Outlook on home PC 710. The communication of the change information to Microsoft Outlook and subsequent updates to the contacts in Outlook are coordinated by the client software on home PC 710. Thus, the contact information in home PC 710 and work PC 702 is again synchronized.

Other transactions, in addition to "Add," are provided with exemplary embodiments of the present invention. One of these is the transaction, "Modify."

Using the present example, after CONT.D001 is uploaded to network 700, the contact information for a person sometimes change. For instance, John Smith may call the user on the telephone and tells the user that John has changed his phone number. The user then accesses office PC 710, changes the phone number for John Smith in the user's contacts.

The user then activates, for example, a "synchronize" button displayed on the computer screen by the client software, so a new data package or the change log, CONT.D002, is created and uploaded to network 700 and stored on one of storage servers 300_1, 300_2. A signal is sent by office PC 710 to management server 1802 informing management server 1802 that data package CONT.D002 has been uploaded. Data package CONT.D002 differs from data packages CONT.D000 and CONT.D001, in that the action, "Modify" is used instead of "Add." The Modify command and the associated change information is correlated with the particular user. In particular, the Modify instruction is associated with the pertinent ID, in this case ID #2 representing John Smith. In addition, data package CONT.D002 includes the field to be modified, in this example, "Phone," and the new information, in this example, John Smith's new phone number.

Subsequently, when home PC 702 connects to network 700, using techniques described above, the data package CONT.D002 is downloaded to home PC 702, and the client software recognizes that, for ID #2, the information within the field "Phone" has been updated. The next time home PC couples to network 700, home PC 702 sends a signal to management server indicating that home PC 702 has received CONT.D002. The modification is then made to this contact information via Microsoft Outlook. The home PC 702 then sends a confirmation signal to management server 1802, confirming that home PC 702 has received and applied the change information in version #2 of the contacts data packages. The pertinent information in the registry for home PC 702 is then updated. If no subsequent data packages with change information for contacts have been stored on the storage servers, then no data packages are downloaded to home PC 702.

As changes are made for various classes of data, data packages accumulate

on the storage servers 300_1, 300_2 and consume storage space. As the number of stored data packages increases, the amount of available storage space on the storage servers decreases. In the example above, data package CONT.D000 occupies 2 kilobytes ("K"), CONT.D001 occupies 1 K, and CONT.D002 occupies 0.5 K. Thus, a total of 3.5 K of storage space on the storage servers is occupied by these three files. In situations where storage space is limited, for example, to 25 megabytes ("M"), a restriction sometimes imposed on a user's account, the amount of available storage space continues to decrease as information is updated, until storage space is no longer available for the user on the storage servers. The user may then become frustrated and generally dissatisfied with the entire data transfer and synchronization system because he can no longer store change logs.

Those both skilled and unskilled in the art will appreciate the user's frustration in the following scenario. In this example, a user has 2000 e-mails in his "In Box" of Microsoft Outlook on his home PC 702. The user desires to synchronize all of his other devices, such as office PC 710, with home PC 702. Thus, a data package MAIL.D000 is created by the client software on home PC 702 and uploaded to network 700 for storage. The data package includes all 2000 e-mails, each having an associated ID# and an associated action "Add." In this example, the data package MAIL.D000 occupies approximately 10 M of memory, for a user who has a total of 25 M allotted to his account. Recognizing this, the user issues a delete command to delete 1500 of the 2000 messages, in order to reduce the amount of occupied space. A new data package MAIL.D001 is then created, containing 1500 "Delete" actions, each associated with a particular one of the e-mail ID #'s in data package MAIL.D000. The new data package MAIL.D001 occupies an additional 1 M of memory, resulting in the occupation of even more storage space on the servers. Consequently, while the user in all events expects the amount of occupied storage space to be reduced from 10 M to about 1/4 of this value, or 2.5 M, the amount actually increases to 11 M.

It is therefore desirable to collapse data packages stored on the storage

servers whenever possible. Collapsing the data packages, as provided in accordance with exemplary embodiments of the present invention, generally entails combining the data packages for a particular class of data, with superfluous information being deleted. Using the example above, data packages MAIL.D000 and MAIL.D001 are combined such that the "Delete" actions in MAIL.D001 replace the "Adds" for the same ID #s in previous data package MAIL.D000, to define a new data package MAIL.D001. In an alternative example, the "Delete" command only applies to one or more fields in a given transaction. This new data package preferably overwrites the original MAIL.D001, that is, the one with only "Delete" actions. Data package MAIL.D000 is no longer relevant and, therefore, is deleted. After "rolling the base" in this fashion, only the new MAIL.D001 data package remains on the storage server, and the amount of occupied storage space is thus reduced from 10 M to approximately 2.5 M, as the user had expected.

In Fig. 18, a base rolling engine 1806, constructed in accordance with an exemplary embodiment of the present invention, is provided to achieve the desired collapsing of data packages. The base rolling engine 1806 is desirably situated within one of the device engines 1808 coupled to network 700. In one exemplary embodiment, the activation of base rolling engine 1806 is controlled by the user, while in another exemplary embodiment, the base rolling engine is activated automatically by the data transfer and synchronization system. In embodiments where the base rolling engine is manually activated, the user accesses his account via one of his devices such as home PC 702, and issues a command to compact his data. In one example, this command is executed by the user simply moving a pointer over a "Roll the Base" button displayed as part of a graphical interface on the user's display screen, and clicking on this button using a controlling device such as a mouse or trackball.

Fig. 19 is a diagram illustrating a collapsing of data packages, performed in accordance with an exemplary embodiment of the present invention. As described above, each data package or change log is stored on a repository such as storage servers 300_1, 300_2. Within each change log are a plurality of items including, in one

example, a Parent ID, an ID, an Action, and one or more Fields. The Parent ID identifies the relationship of a particular item with another item, for example, in situations where the items are related hierarchically. The ID and Action items are defined above. The fields in each data package identify what particular fields, for the class of data, are to be changed. Each field preferably has a unique numeric value and includes an attribute representing change information for the field.

In Fig. 19, there are two data packages, a base version D000 and a subsequent version D001. Both versions include the items described above and one or more fields. In particular, base version D000 has an ID of 2, an Action "Add" and three fields: "FirstName," which has the attribute "John," "LastName," having the attribute "Smith," and "Web Page," having an associated URL, "http::// . . . " Version D002 also has an ID of 2, but a change action of "Modify" rather than "Add." Version D001 only has one field, "FirstName" which has the attribute "Scott."

In Fig. 19, when the user issues a command to "roll the base," the base rolling engine determines whether version D000 and D002 have one or more of the same ID #s. In this example, because both D000 and D002 have the same ID #2, the two data packages are collapsed into one file. Specifically, a new version D002 is created, replacing the original D002 data package. The Parent ID and ID # of 2 remain the same, and the change information from D002 is applied to version D000. Specifically, the field "FirstName" that both D000 and D002 have in common is identified, and assigned the more recent attribute, "Scott" from data package D002. The Action remains "Add," and the fields "LastName" and "Web Page" remain as "Smith" and URL, "http:// . . .," respectively, from version D000. Thus, in this example, the new D002 is essentially the same as D000, except that the field "FirstName" has been replaced with the attribute "Scott." The modification contained in original data package D002 has actually been made to data package D000, resulting in new data package D0002. The data package D000 is then deleted.

In another exemplary embodiment, shown in Fig. 20, three devices are capable of coupling to data network 700, namely Device A, Device B, and Device C. In one example, Device A is a palmtop computer, Device B is a home PC, and

Device C is an office PC. In other examples, various other devices as described above are used for Devices A, B and C. In Fig. 20, although at least 15 different change logs for contact information have been uploaded to the data network, Devices A, B and C each have different versions of contacts. The contacts in Device A have only been updated to include changes in CONT_D003, Device B is updated to incorporate CONT.D010, and the Device C has been updated to CONT.D015. In this example, D0XX represents sequential versions of contact information; the first change log in the sequence is CONT.D000, and the 15th change log in the sequence is CONT.D015. All of these change logs are stored on storage servers coupled to the data network.

In Fig. 20, although the versions of contact information among the various devices are spread apart, that is, not in sequence, the 15 data packages stored on the storage servers are collapsed according to exemplary embodiments of the present invention. First, a plurality of bases versions are defined. In this example, the first base data package is defined by collapsing sequential data packages, as described above, starting with CONT.D000 through the version of Device A, in this example, CONT.D003. The second base data package is defined starting with version CONT.D003 of Device A, and collapsing sequential change logs through the version of Device B, in this example, CONT.D010. The third base data package is similarly defined by collapsing sequential data packages between CONT.D010 and CONT.D015. This results in three sequential base data packages, which replace data packages CONT.D013, D014, and D015. Device A is then updated to include changes to contact information up to and including new CONT.D013, Device B to CONT.D014, and Device C to version D015.

Throughout the process of defining the new base data packages, management server 1802 maintains in the registry for each device the most current version # of contact information stored in that device. In the example above, before the collapsing operating, Device A is at version 3, Device B at version 10, and Device C at version 15. The base rolling engine is in communication with management server 1802, so that when the data packages are collapsed, the base

rolling engine requests and receives device information for the particular user from management server 1802. This includes information identifying all of the devices registered by the user, and what is the most current version of the data class stored in each device.

Thus, after the collapsing operation, when Device B is to be synchronized, management server recognizes that Device B is at version 14, so only data package CONT.D015 needs to be downloaded to Device B. Similarly, when Device A couples to the network to be synchronized, the change information in CONT.D014 is first applied, then CONT.D015. These updates are all achieved using the techniques described above.

Performing one or two updates during the time a device couples to the network is computationally less complex and, therefore, faster than performing an entire sequence of updates. This is due to the fact that for every file that is downloaded, a communications channel must be established, the file downloaded, and then the channel closed. There is high overhead associated with this opening and closing connections. The more times this iteration is performed, the more time is monopolized, resulting in higher costs. In the example above, without rolling the base as described to bring Device A current to version #13 information, 12 updates would need to be performed to update Device A from version #3 to version #15. Using the example above, with 12 files, a connection must be opened, file downloaded, and connection closed, 12 times. With 3 files, this iteration need only be performed 3 times. Fewer data packages are sent from the network to Device A and, therefore, less data. This, in turn, results in less processing by Device A and improved efficiency.

The aforementioned exemplary embodiments of the present invention provide a user-centric model of communication to deliver personal information via network services. This model accommodates devices that are disconnected from the network, such as the Internet, at various times. Personal information can continue to exist locally rather than imposing a server-centric model on existing information.

In accordance with the foregoing, a store and forward information broadcast is

utilized. Changes to existing information are replicated to an Internet storage server and changes are then retrieved by other devices on the network at device-specific times. In this manner, direct client communication is accomplished without requiring one-to-one communication. While one communication is supported by the system of the present invention, it need not be required.

Although the present invention has been presented in the form of an Internet store and forward broadcast for the purposes of synchronizing personal information amongst various types of devices, it will be readily recognized that synchronization need not be accomplished as the only application for the aforementioned system. In particular, the system can be utilized to efficiently broadcast changes to information in so-called "push" type information applications where only portions of the data need to be changed on a client application. For example, in a system where information such as changes in a stock price need to be broadcast to a plurality of users, a client application implementing the aforementioned technology can be updated by only changing specific portions of the data in the client application relative to that particular stock price. This can be done using a smaller bandwidth than has previously been determined with other devices.

It should be understood that the exemplary embodiments described above are only illustrative of the principles of the present invention. Additional variations will be apparent to those skilled in the art and, therefore, can be made without departing from the scope and spirit of the invention. Thus, the invention is not limited to the particular details described above. Rather, it is intended that the claims below cover all such variations and modifications as are within the scope and spirit of the invention.

CLAIMS

What is claimed is:

1. A method of collapsing data packages stored in a data transfer and synchronization system, the method comprising:

providing a first data package having a first transaction including an identification number, an action, and a plurality of fields each with an attribute representing change information;

providing a second data package having a second transaction made subsequent to the first transaction, the second transaction having an identification number, an action, and a field with an attribute;

determining whether the identification number of the second transaction corresponds to the identification number of the first transaction;

determining whether the field of the second transaction corresponds to one of the fields of the first transaction;

combining, when the identification numbers of the first and second transactions correspond to one another and the field of the second transaction corresponds to one of the fields of the first transaction, the first and second data packages to define a combined data package having a combined transaction with the identification number; and

replacing the second data package with the combined data package.

2. The method of claim 1 further comprising:

deleting the first data package.

3. The method of collapsing data packages of claim 1, wherein combining the first and second data packages comprises:

determining the type of action of the second transaction;

defining, when the action of the second transaction is "Add," the combined transaction to include an "Add" action and the corresponding field and the attribute of

the second transaction;

defining, when the action of the second transaction is "modify," the combined transaction to include an "add" action and the corresponding field and the attribute of the second transaction; and

defining, when the action of the second transaction is "delete," the combined transaction to include a "delete" action and the corresponding field.

4. A method of collapsing data packages stored in a data transfer and synchronization system, the method comprising:

providing a first data package having a plurality of first transactions each including an identification number, an action, and a plurality of fields each with an attribute representing change information;

providing a second data package having a second transaction made subsequent to the first transactions, the second transaction having an identification number, an action, and a field with an attribute;

determining whether the identification number of the second transaction corresponds to one of the identification numbers of the first transactions;

identifying, when the identification number of the second transaction corresponds to the one of the identification numbers of the first transactions, the one first transaction;

determining whether the field of the second transaction corresponds to one of the fields of the identified first transaction;

combining, when the identification numbers of the second transaction and the identified first transaction correspond to one another and the field of the second transaction corresponds to one of the fields of the identified first transaction, the first and second data packages to define a combined data package having a combined transaction with the identification number; and

replacing the second data package with the combined data package.

5. A method of collapsing data packages stored in a data transfer and

synchronization system, the method comprising:

collapsing a first plurality of data packages to define a first base data package associated with a first device, each data package having a transaction, all of the transactions having been applied to data in the first device;

collapsing a second plurality of data packages to define a second base data package associated with a second device, each data package having a transaction, all of the transactions having been applied to data in the second device; and

collapsing a third plurality of data packages to define a third base data package associated with a third device, each data package having a transaction, all of the transactions having been applied to data in the third device.

6. A base rolling apparatus for collapsing data packages stored in a data transfer and synchronization system, the base rolling apparatus situated in a device engine on a server coupled to a data network, the apparatus comprising:

a first providing part which provides a first data package having a first transaction including an identification number, an action, and a plurality of fields each with an attribute representing change information;

a second providing part which provides a second data package having a second transaction made subsequent to the first transaction, the second transaction having an identification number, an action, and a field with an attribute;

a first determining part which determines whether the identification number of the first transaction corresponds to the identification number of the second transaction;

a second determining part which determines whether the field of the second transaction corresponds to one of the fields of the first transaction;

a third combining part which combines, when the identification numbers of the first and second transactions correspond to one another and the field of the second transaction corresponds to one of the fields of the first transaction, the first and second data packages to define a combined data package having a combined transaction with the identification number; and

a replacing part which replaces the second data package with the combined data package.

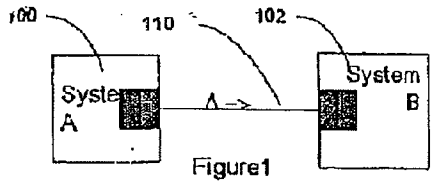
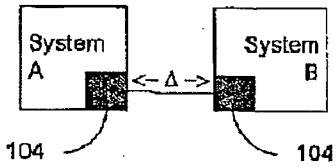


Figure1

Figure 2



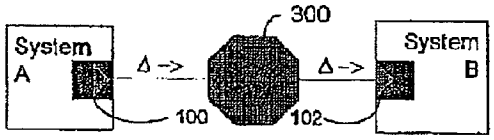


Figure 3

Figure 4

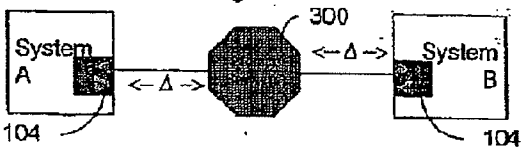


Figure 5

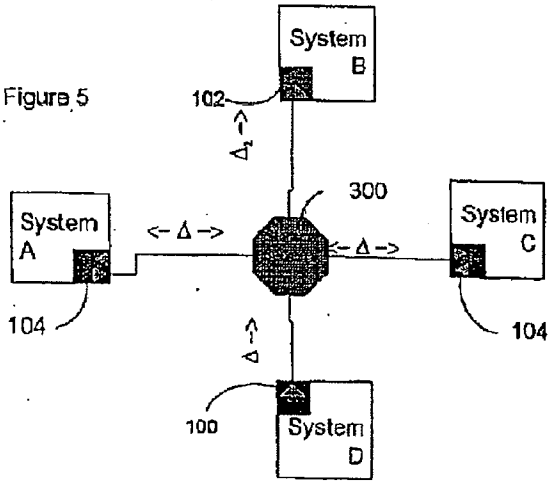


Figure 6

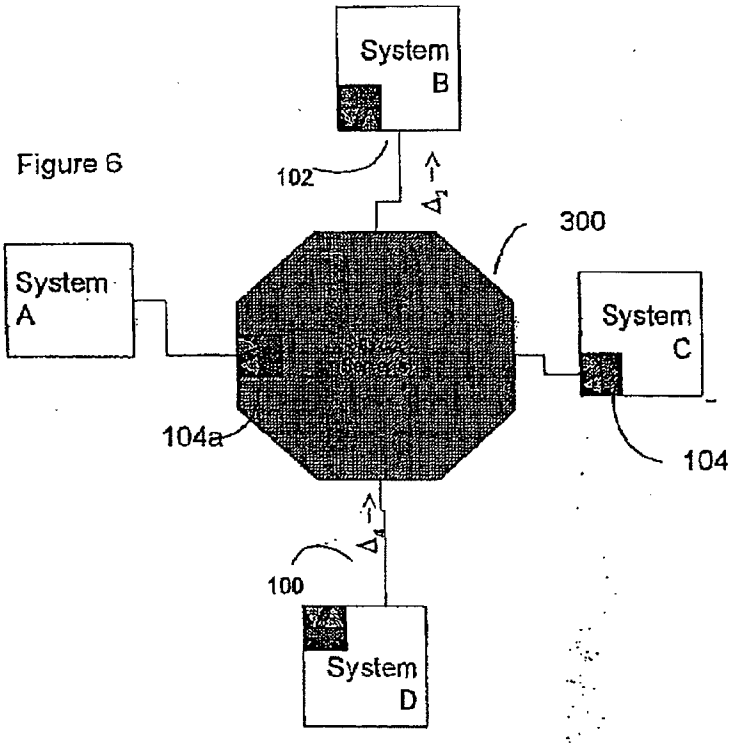


Figure 7

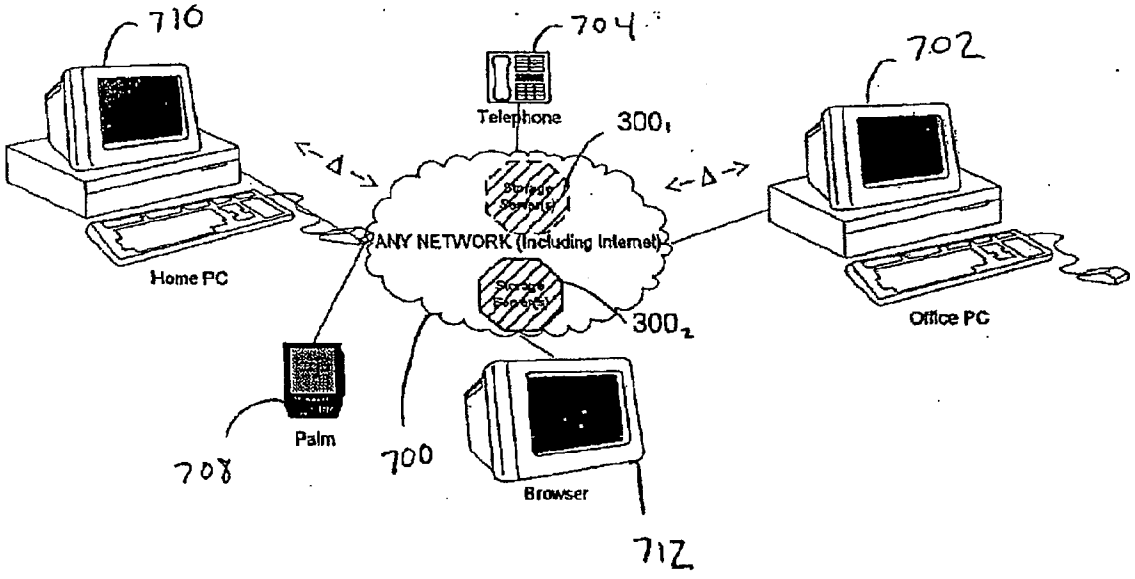


Figure 8

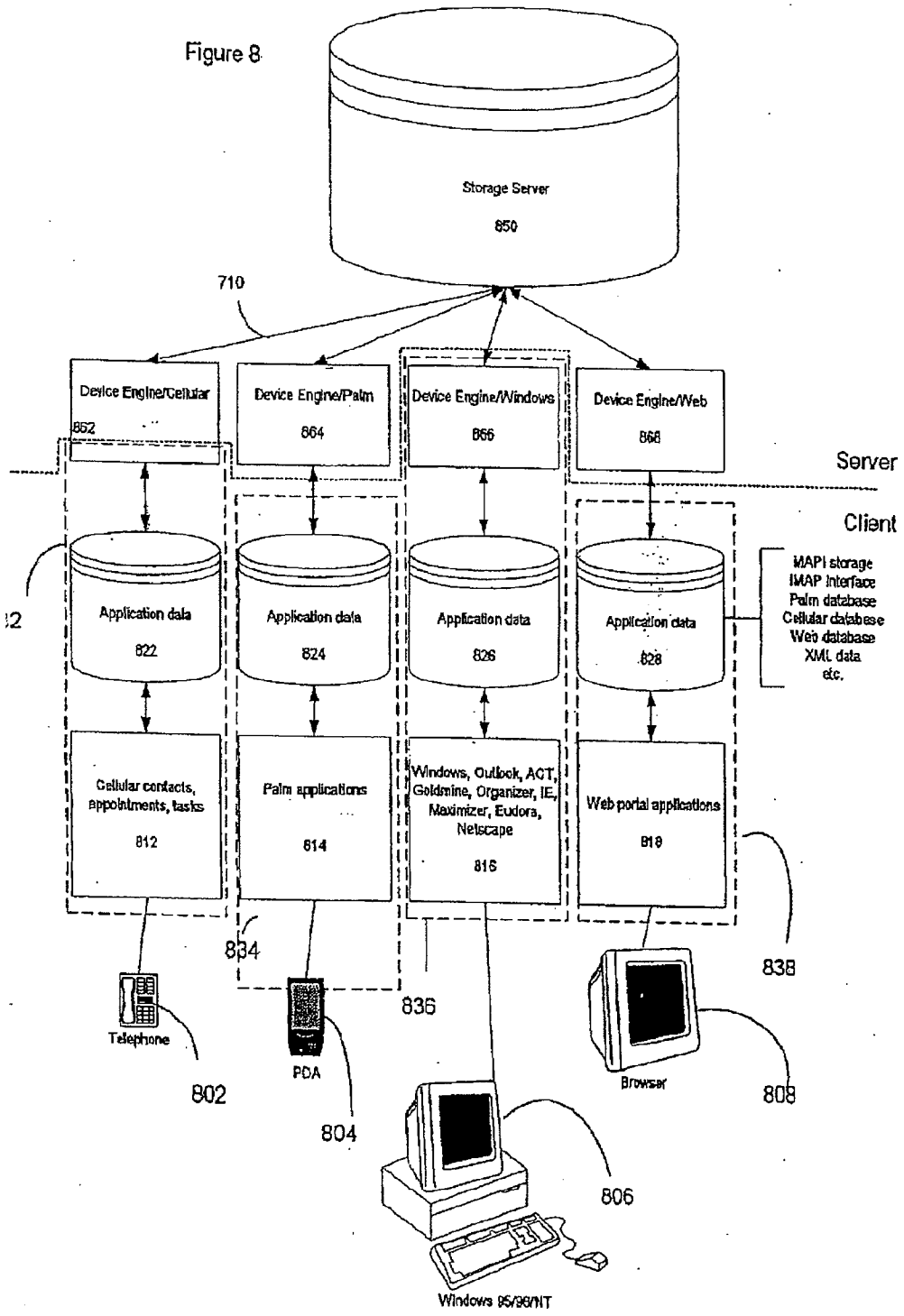

Storage Server
850
710
Device Engine/Cellular
852
Device Engine/Palm
864
Device Engine/Windows
866
Device Engine/Web
868
Server
Client
MAPI storage
IMAP interface
Palm database
Cellular database
Web database
XML data
etc.
Application data
822
Application data
824
Application data
826
Application data
828
Cellular contacts, appointments, tasks
812
Palm applications
814
Windows, Outlook, ACT, Goldmine, Organizer, IE, Maximizer, Eudora, Netscape
816
Web portal applications
818
834
836
838
Telephone
802
PDA
804
806
Browser
808
Windows 95/98/NT

Figure 9A
Desktop
Device Engine

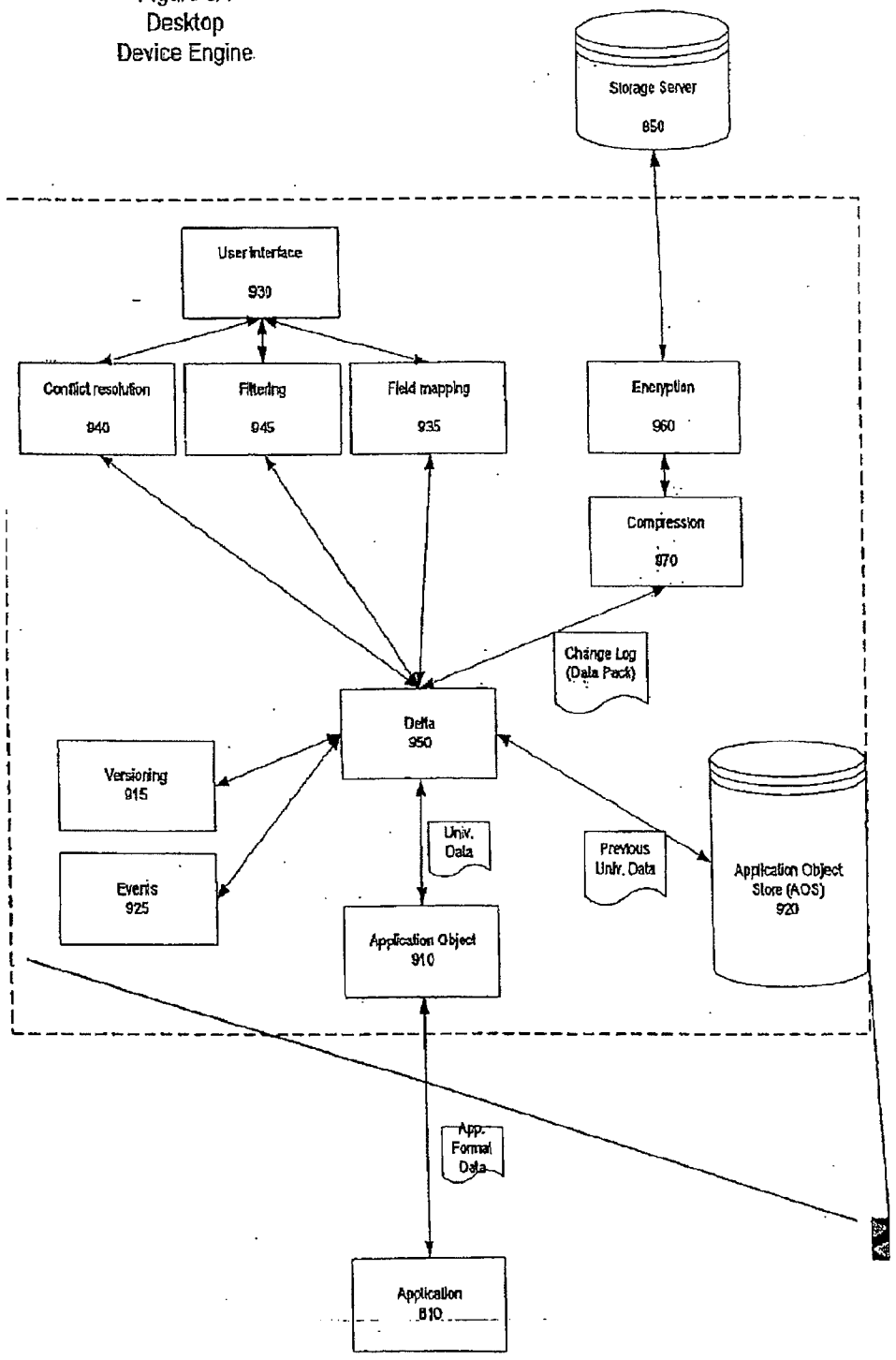
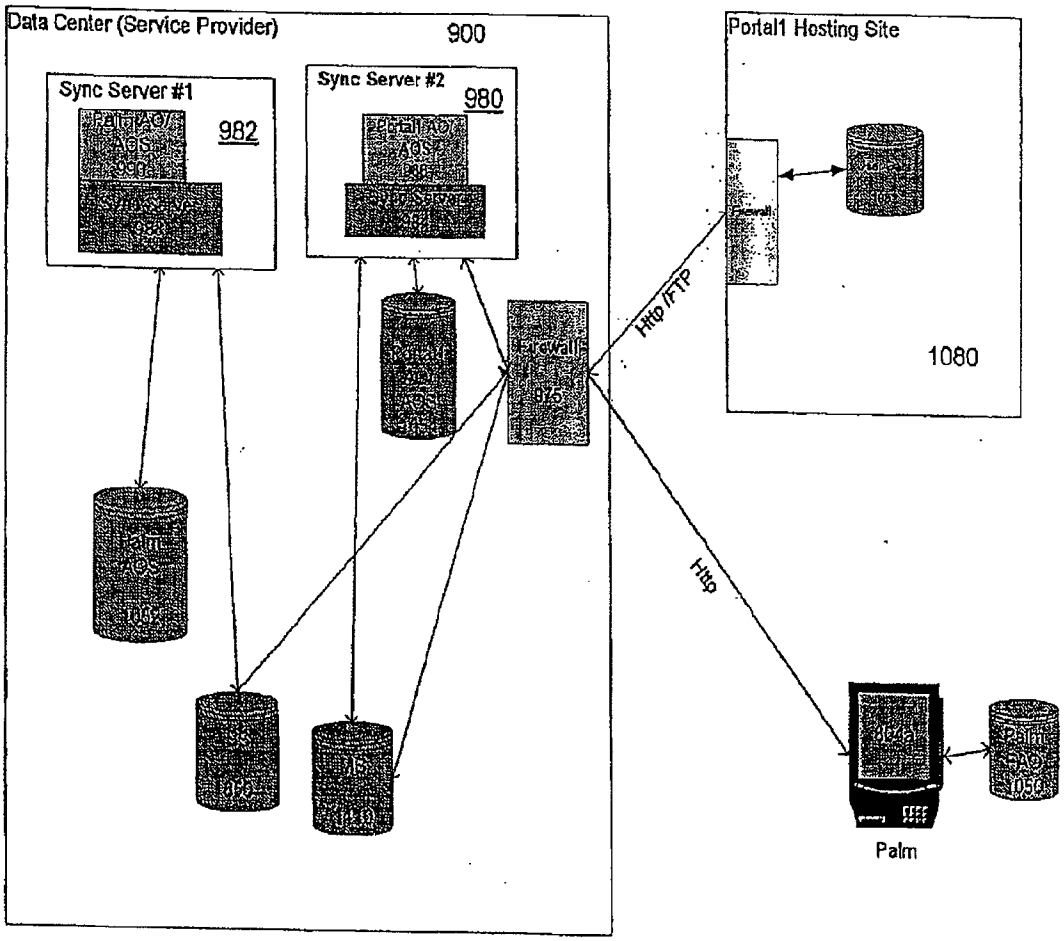


Figure 9B


Data Center (Service Provider)
900
Sync Server #1
982
Sync Server #2
980
Firewall
975
Http /FTP
Portal1 Hosting Site
1080
Http
Palm

Device Engine/Windows

Figure 10

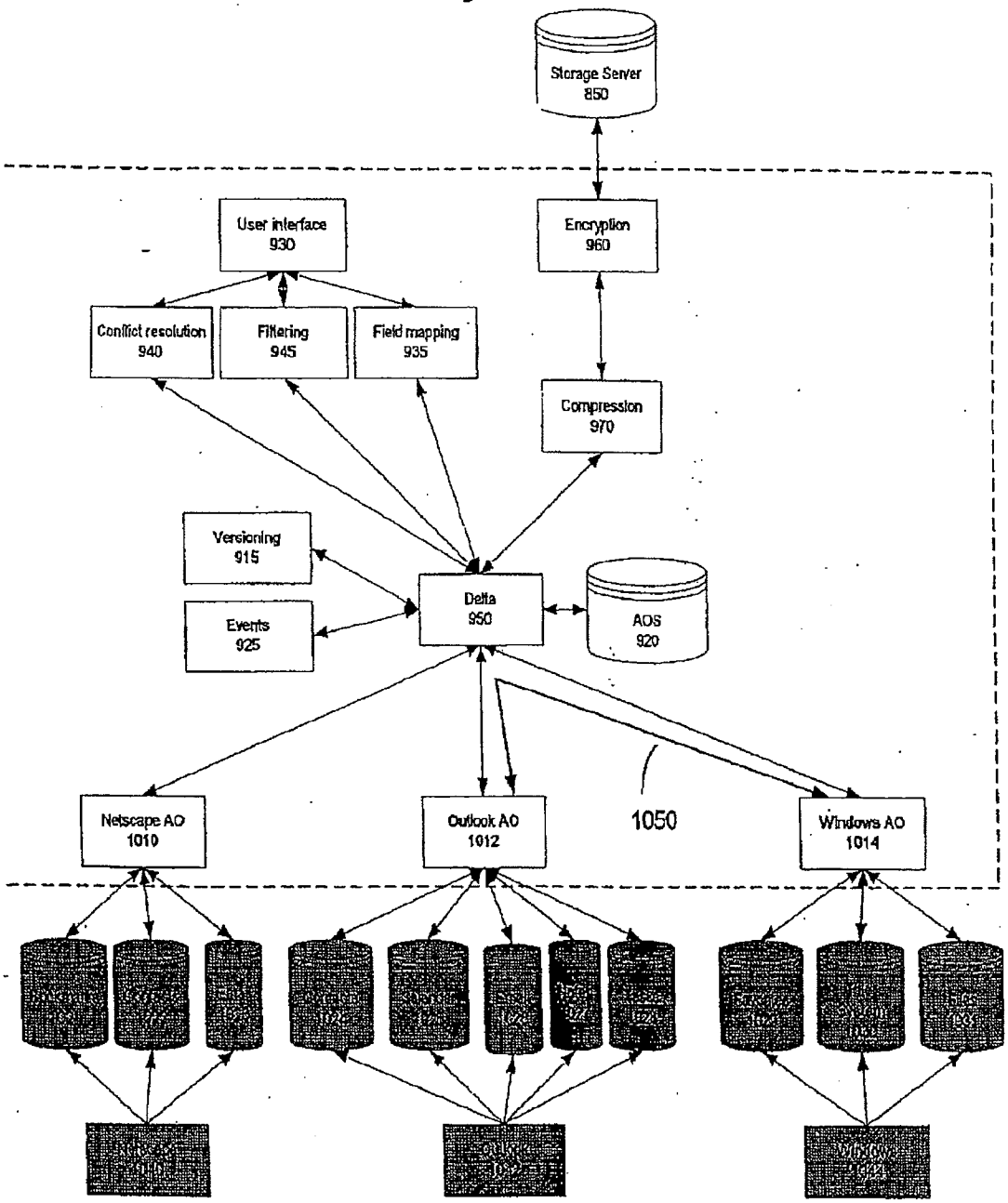


Figure 11

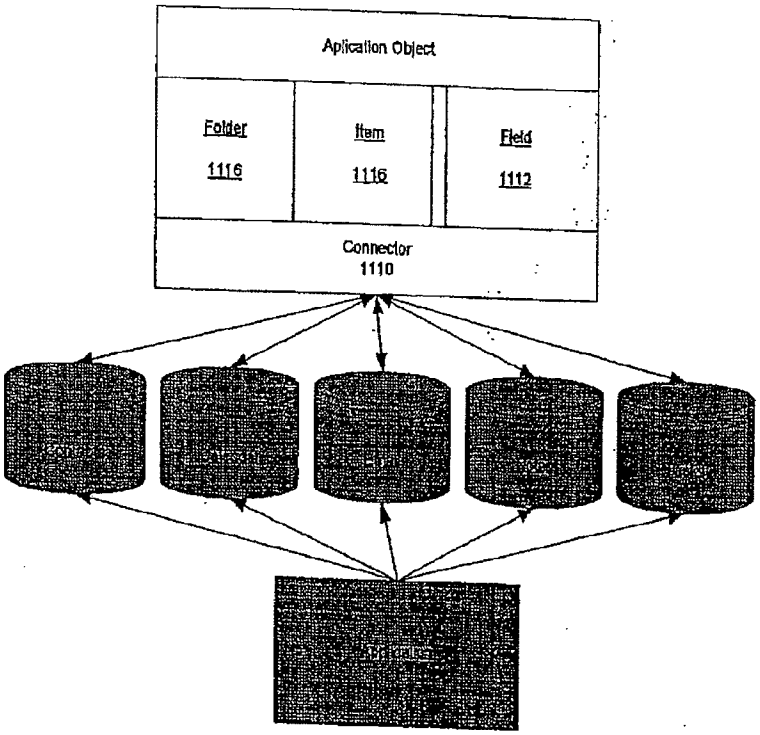

Application Object
Folder
1116
Item
1116
Field
1112
Connector
1110

Figure 12
Object Hierarchy

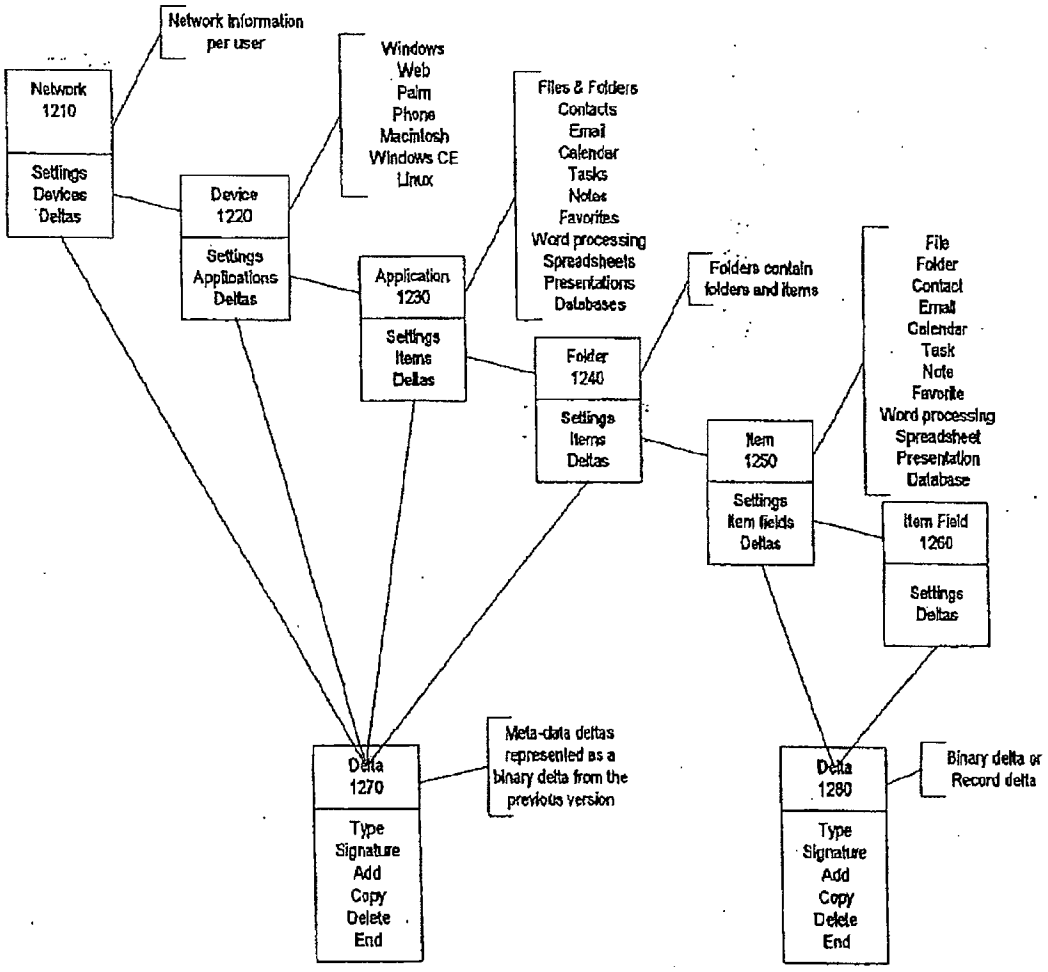


Figure 13

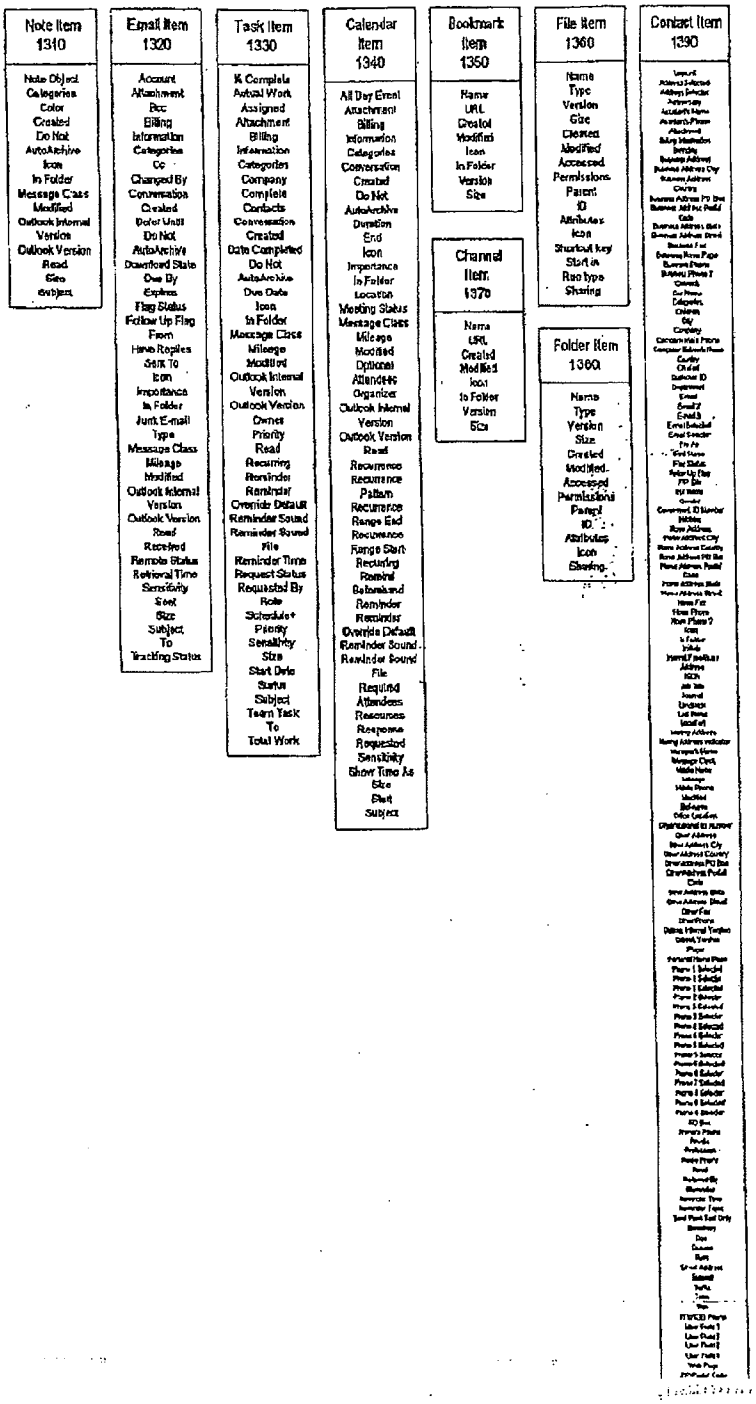

Note Item 1310
Note Object
Categories
Color
Created
Do Not AutoArchive
Icon
In Folder
Message Class
Modified
Outlook Internal Version
Outlook Version
Read
Size
Subject
Email Item 1320
Account
Attachment
Bcc
Billing Information
Categories
Cc
Changed By
Conversation
Created
Defer Until
Do Not AutoArchive
Download State
Due By
Expires
Flag Status
Follow Up Flag
From
Have Replies
Sent To
Icon
Importance
In Folder
Junk E-mail Type
Message Class
Mileage
Modified
Outlook Internal Version
Outlook Version
Read
Received
Remote Status
Retrieval Time
Sensitivity
Sent
Size
Subject
To
Tracking Status
Task Item 1330
% Complete
Actual Work
Assigned
Attachment
Billing Information
Categories
Company
Complete
Contacts
Conversation
Created
Date Completed
Do Not AutoArchive
Due Date
Icon
In Folder
Message Class
Mileage
Modified
Outlook Internal Version
Outlook Version
Owner
Priority
Read
Recurring
Reminder
Reminder Override Default
Reminder Sound
Reminder Sound File
Reminder Time
Request Status
Requested By Role
Schedule+ Priority
Sensitivity
Size
Start Date
Status
Subject
Team Task
To
Total Work
Calendar Item 1340
All Day Event
Attachment
Billing Information
Categories
Conversation
Created
Do Not AutoArchive
Duration
End
Icon
Importance
In Folder
Location
Meeting Status
Message Class
Mileage
Modified
Optional Attendees
Organizer
Outlook Internal Version
Outlook Version
Read
Recurrence
Recurrence Pattern
Recurrence Range End
Recurrence Range Start
Recurring
Reminder
Reminder Before
Reminder Override Default
Reminder Sound
Reminder Sound File
Required Attendees
Resources
Response Requested
Sensitivity
Show Time As
Size
Start
Subject
Bookmark Item 1350
Name
URL
Created
Modified
Icon
In Folder
Version
Size
Channel Item 1370
Name
URL
Created
Modified
Icon
In Folder
Version
Size
File Item 1360
Name
Type
Version
Size
Created
Modified
Accessed
Permissions
Parent ID
Attributes
Icon
Shortcut key
Start in
Run type
Sharing
Folder Item 1360
Name
Type
Version
Size
Created
Modified
Accessed
Permissions
Parent ID
Attributes
Icon
Sharing
Contact Item 1390

Figure 14

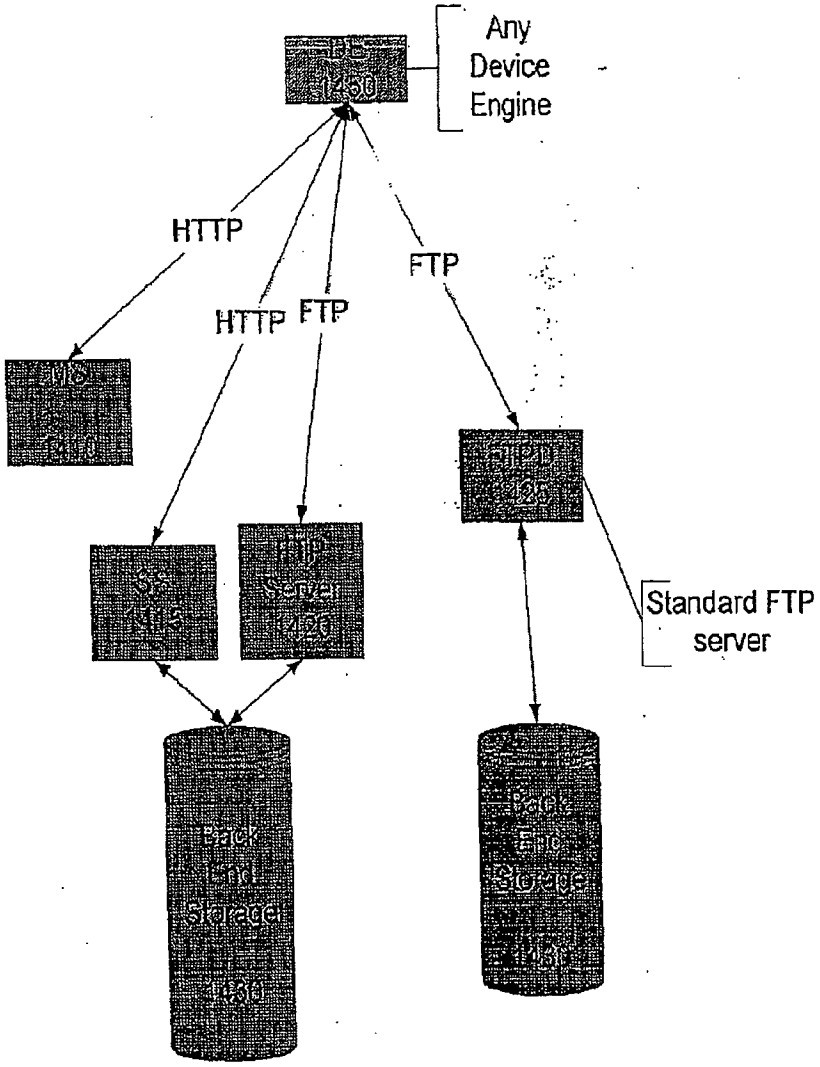


Figure 15
Pull Synchronization

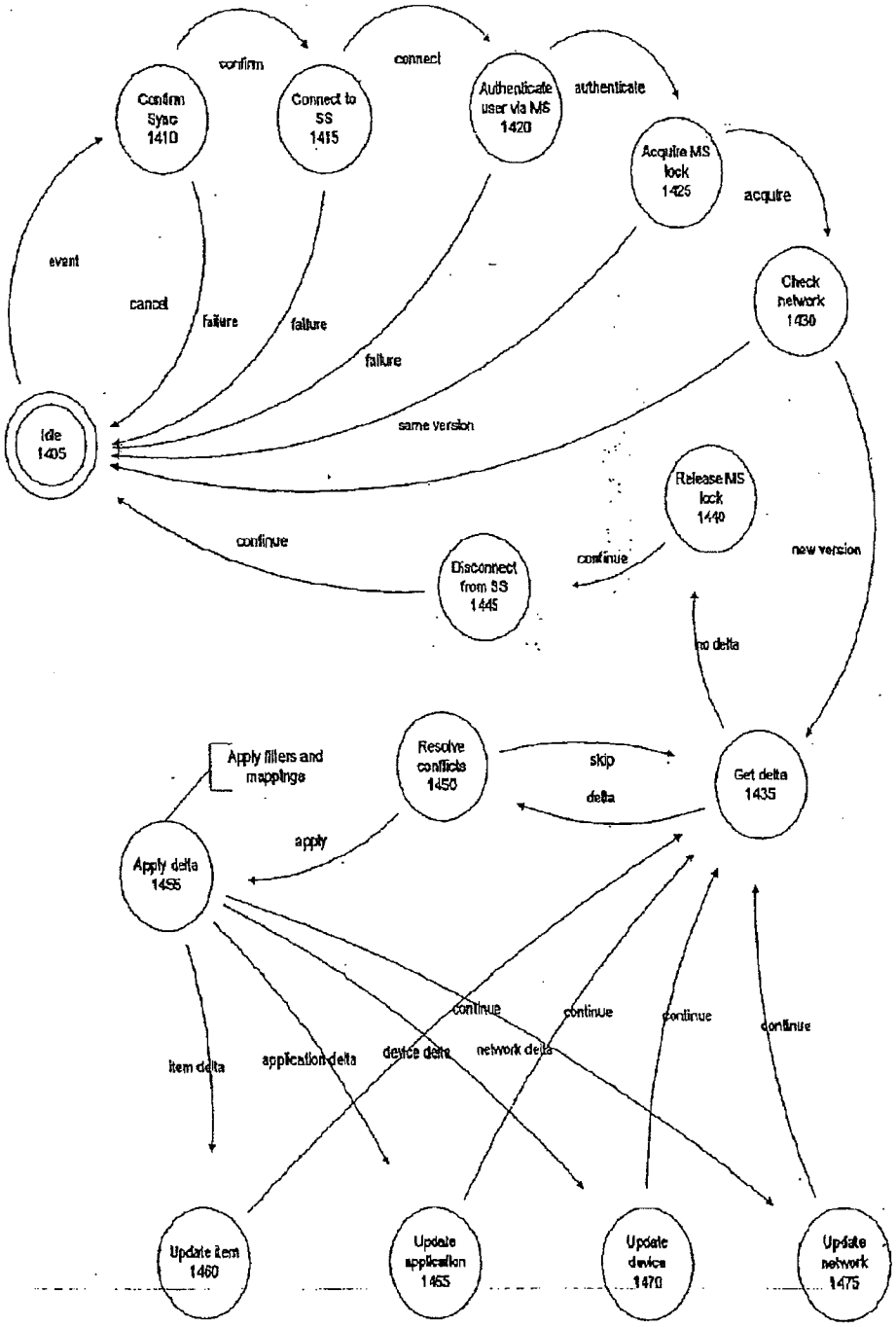


Figure 16
Push Synchronization

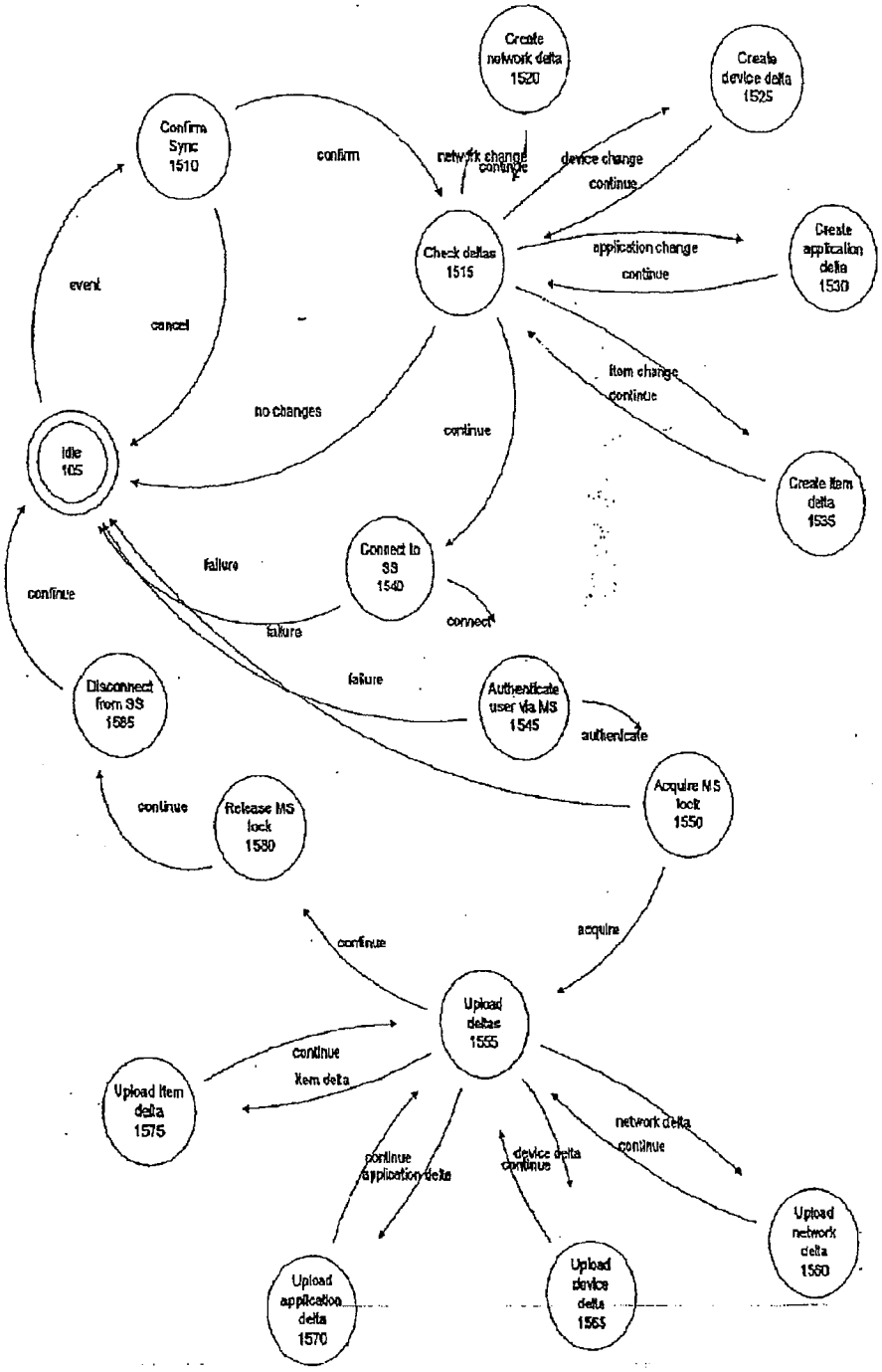
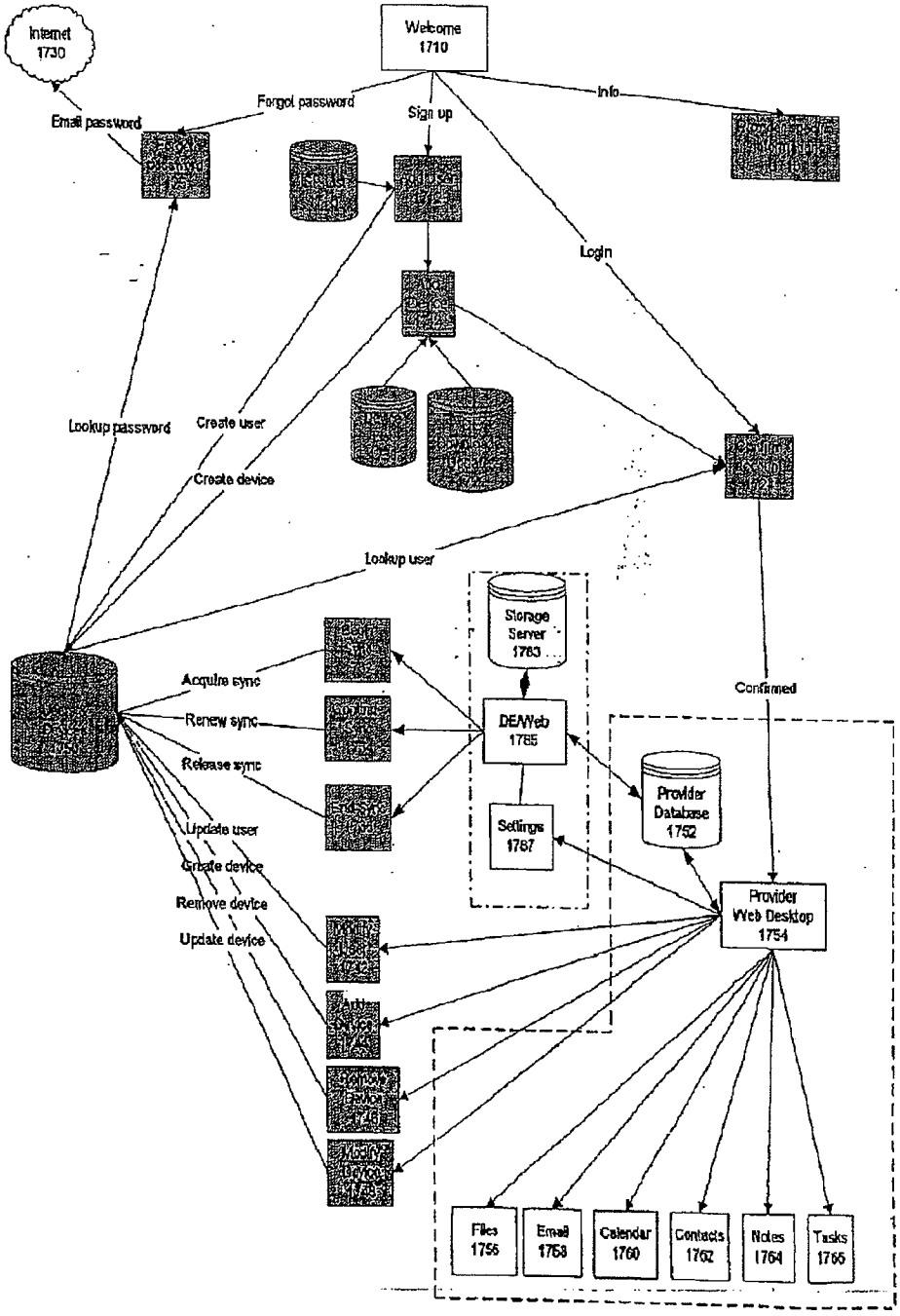


Figure 17


Internet 1730
Welcome 1710
Email password
Forgot password
Sign up
Info
Login
Lookup password
Create user
Create device
Lookup user
Storage Server 1763
Acquire sync
Renew sync
Release sync
Update user
Create device
Remove device
Update device
DE/Web 1765
Settings 1767
Provider Database 1752
Confirmed
Provider Web Desktop 1754
Files 1756
Email 1758
Calendar 1760
Contacts 1762
Notes 1764
Tasks 1766

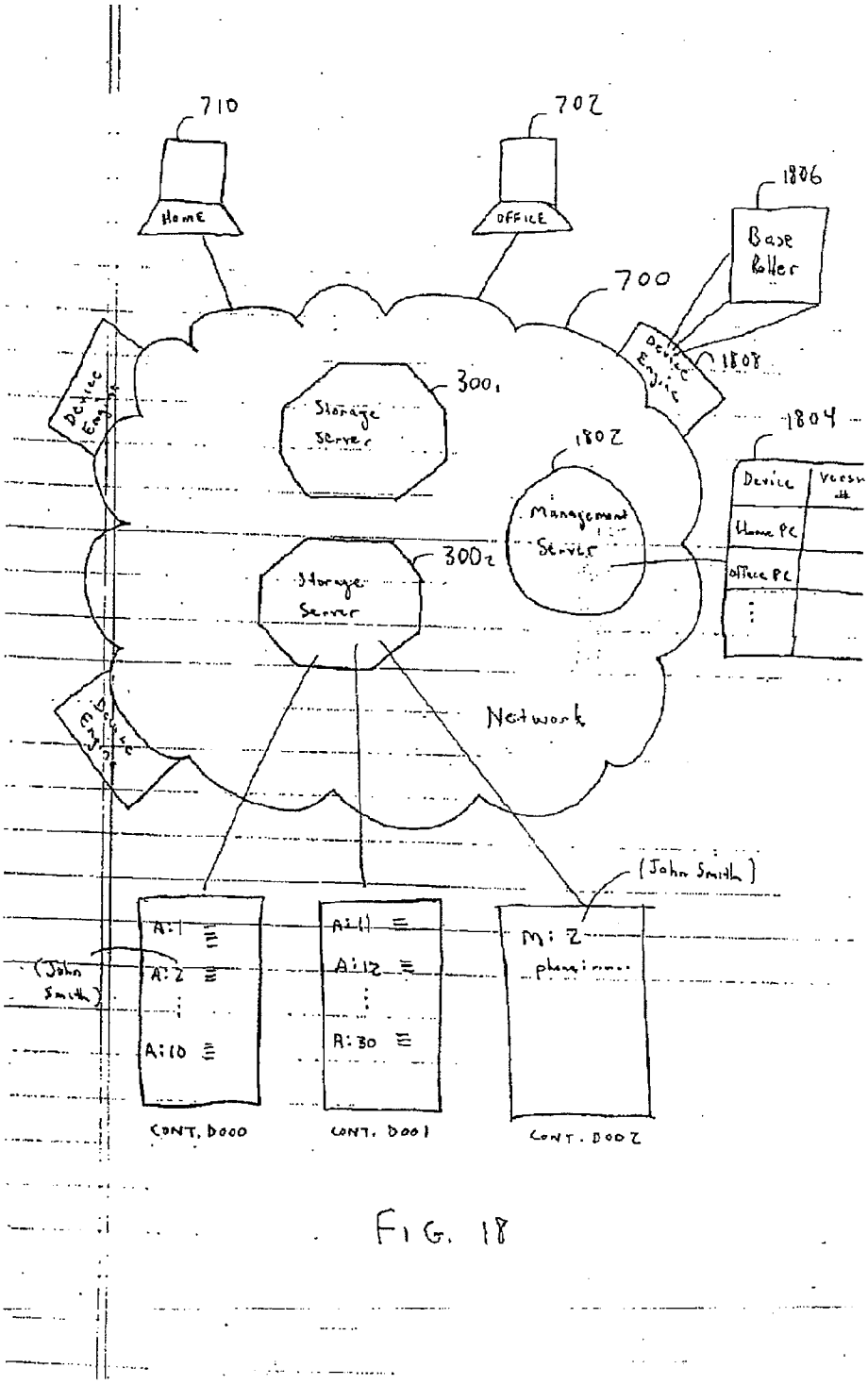

710
Home
702
OFFICE
1806
Base Roller
700
Device Engine
1808
Storage Server
300₁
1804
Device
Management Server
180Z
Storage Server
300₂
Network
(John Smith)
A:1 ≡
A:2 ≡
A:10 ≡
A:11 ≡
A:12 ≡
A:30 ≡
M:Z
CONT. D000
CONT. D001
CONT. D00Z

FIG. 18

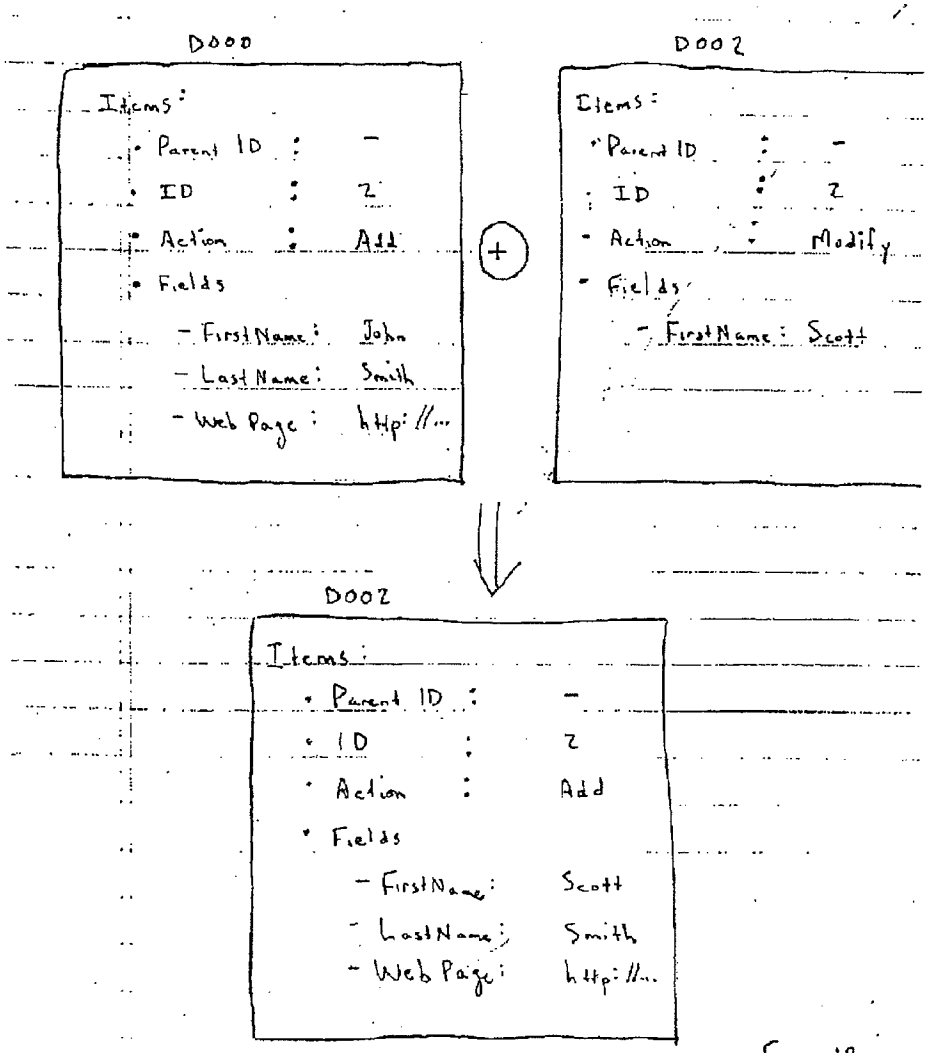

D000
Items:
• Parent ID : -
• ID : 2
• Action : Add
• Fields
- FirstName: John
- LastName: Smith
- Web Page : http://...
+
D002
Items:
• Parent ID : -
• ID : 2
• Action : Modify
• Fields
- FirstName: Scott
D002
Items:
• Parent ID : -
• ID : 2
• Action : Add
• Fields
- FirstName: Scott
- LastName: Smith
- Web Page: http://...

FIG. 19

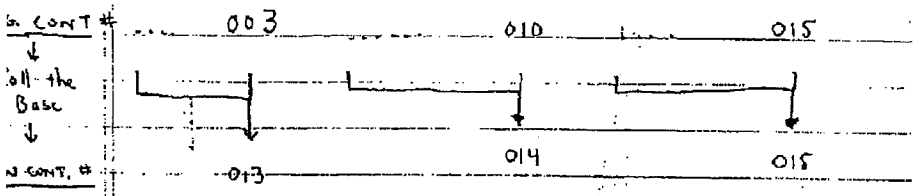


FIG. 20

ABSTRACT

A base rolling engine for collapsing data packages stored in a data transfer and synchronization system. A first data package is provided. The first data package has a first transaction including an identification number, an action, and a plurality of fields. Each field has an attribute representing change information. A second data package is also provided. The second data package has a second transaction made subsequent to the first transaction. The second transaction has an identification number, an action, and a field with an attribute. The base rolling engine determines whether the identification number of the second transaction corresponds to the identification number of the first transaction. The base rolling engine also determines whether the field of the second transaction corresponds to one of the fields of the first transaction. When the identification numbers of the first and second transactions correspond to one another, and the field of the second transaction corresponds to one of the fields of the first transaction, the first and second data packages are combined. A combined data package is thus defined having a combined transaction with the identification number. The combined data package replaces the second data package, and the first data package is deleted.